VARCHART
JGantt

JavaBeans Edition
Version 3.0
Benutzerhandbuch

NETRONIC

VARCHART JGantt

JavaBeans

Version 3.0

Benutzerhandbuch

NETRONIC Software GmbH
Pascalstraße 15
52076 Aachen
Deutschland
Tel: +49 (0) 2408 141-0
Fax: +49 (0) 2408 141-33
E-Mail sales@netronic.de
www.netronic.de

© Copyright 2012 NETRONIC Software GmbH
Alle Rechte vorbehalten.

Die Informationen im vorliegenden Benutzerhandbuch werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht. Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt. Bei der Zusammenstellung von Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen. Trotzdem können Fehler nicht vollständig ausgeschlossen werden. Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

Das illegale Kopieren und Vertreiben dieses Produktes stellt einen Diebstahl geistigen Eigentums dar und wird von NETRONIC Software GmbH strafrechtlich verfolgt.

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien. Die gewerbliche Nutzung der in diesem Buch gezeigten Abbildungen ist nicht zulässig.

Microsoft Windows, Microsoft Explorer, Microsoft Visual Basic und Microsoft Visual Studio sind Warenzeichen der MICROSOFT Corp.

Bearbeitungsstand: 21. Mai 2012

Inhaltsverzeichnis

1 Einleitung 7

1.1 Was ist VARCHART JGantt? 7
1.2 Der Aufbau eines Gantt-Diagramms 8
1.3 Die VARCHART JGantt Handbücher 10
1.4 Die Installation von VARCHART JGantt 10
1.5 Die Arbeitsweise mit VARCHART JGantt 11
1.5.1 Grundsätzliche Vorgehensweise 11
1.5.2 Nützliche Werkzeuge 12

2 Die Sichtweise des Benutzers 12

3 Ein bestehendes Gantt-Diagramm verändern 16

3.1 Veränderungen an der JGantt-Instanz 16
3.2 Veränderungen an der Zeitskala 17
3.2.1 Anfangs- und Enddatum 17
3.2.2 Zeitskalentyp 18
3.2.3 Auflösung der Zeitskala 18
3.2.4 Position der Zeitskala 19
3.2.5 3D-Effekt 19
3.2.6 Farbgebung 20
3.2.7 Schriftarten 22
3.3 Veränderungen an Knoten 23
3.3.1 Anlage und Auswahl des Entitäten-Sets 23
3.3.2 Knotentermine 26
3.3.3 Die Farbgebung des Knotens 29
3.3.4 Das Design des Knotens 30
3.3.5 Knotensymbole 34
3.3.6 Knoten ohne Ausdehnung 35
3.3.7 Die Knotenbeschriftung 36

3.3.8 Die Positionierungslinie am Knotenphantom 40
3.4 Veränderungen am Gantt-Graphen 40
3.4.1 Die Farbgebung des Gantt-Graphen 40
3.4.2 Die Terminlinie....41
3.5 Veränderungen an der Tabelle 42
3.5.1 Die Festlegung der Tabellenspalten 42
3.5.2 Die Festlegung der Quelle für Spaltenüberschriften 44
3.5.3 Die Farbgebung der Tabelle 44
3.5.4 Festlegung von Schriftarten in der Tabelle 45

4 Gantt-Anwendung neu entwickeln 47

4.1 Automatisch erzeugter Quellcode....47
4.2 Die Lizenzierung des VARCHART JGantt JavaBean-Objektes....51
4.2.1 Arten der Lizenzierung....51
4.2.2 Die Verwendung einer Lizenz....51
4.3 Datenstrukturen....52
4.4 Das Anlegen von Datenstrukturen....53
4.5 Das Anlegen von Daten....57
4.6 Ereignisse der AppData-Komponente 59

5 Die Gestaltung des Gantt-Diagramms.... 63

5.1 Die Gestaltung der Tabelle 63
5.1.1 Aufbau und logische Struktur....63
5.1.2 Gestaltung über Eigenschaften 64
5.1.3 Gestaltung über die Inhaltsdefinition 66
5.1.4 Weitere Tabellenfunktionalität 69
5.2 Zeilenfarben für Tabelle und Gantt-Graph 70
5.3 Knoten im Gantt-Graphen....74
5.3.1 Knotentermine 74
5.3.2 Gestaltung von Knoten 76
5.3.3 Die Sortierung von Knoten....87
5.3.4 Knoten – Gruppierung oder Hierarchie....89
5.3.5 Gruppierung....91
5.3.6 Hierarchische Strukturen bei Knoten 99

5.4 Visualisierung zeitlicher Muster ... 111
5.4.1 Kalender ... 113
5.4.2 Profile ... 114
5.4.3 Bestückung von Profilen ... 117
5.5 Kalenderpersistenz ... 122
5.6 Dynamische Farbgebung ... 123
5.6.1 Dynamische Farbgebung bei Knoten ... 125
5.6.2 Dynamische Farbgebung bei Profilen ... 126
5.7 Erweiterte Optionen der Zeitskala ... 128
5.7.1 Die Formatierung der Zeitskalenbeschriftung ... 128
5.7.2 Dynamik bei Zeitskalentypen ... 132
5.7.3 Zeitskalenabschnitte ... 133
5.8 Verbindungen im Gantt-Graphen ... 135
5.8.1 Interaktionen mit Verbindungen ... 136
5.8.2 Programmierung von Verbindungen ... 138
5.9 Das Histogramm ... 139
5.9.1 Flächen, Kurven und Grenzwertlinien ... 139
5.9.2 Kalender im Histogramm ... 145

6 Interaktionen des Benutzers ... 147
6.1 Konfiguration von Interaktionen ... 147
6.2 Steuerung von Interaktionen über Ereignisse ... 150
6.3 Diagramm-Schaltleiste ... 155
6.4 Aktionen ... 159

7 Zeitberechnungen ... 165
7.1 Möglichkeiten der Zeitberechung ... 165
7.2 Einstellungen bei der Zeitberechung ... 166

8 Das Synchronizer-Panel 169

9 Der Persistenz-Manager 172

10 Der Eigenschaften-Browser 173

11 Der Eigenschaften-Editor 176

11.1 Grundlegende Funktionen 176
11.2 Anwendungsszenarien 178
11.2.1 Das Kennenlernen des Eigenschaften-Editors 178
11.2.2 Die Verwendung selbstgeschriebener Klassen 179
11.2.3 Übertragen von Testeinstellungen auf Produktivdaten 180
11.2.4 Neubeginn einer Applikation 181
11.2.5 Einen Schnappschuss machen 181
11.2.6 Eigenschaften übertragen 182
11.2.7 Eine Alt-Applikation editorfähig machen 183

12 Spracheinstellungen 185

12.1 Erstellung einer Properties-Datei 185
12.2 Verzeichnisse der Properties-Dateien 186

13 Drucken mit VARCHART JGantt 189

13.1 Der Umfang der Druckfunktionalität 189
13.2 Funktionsleisten der Seitenansicht 190

14 Index 205

1 Einleitung

1.1 Was ist VARCHART JGantt?

VARCHART JGantt ist eine JavaBeans Softwarekomponente der Firma NETRONIC, mit der Java-Programmierer grafisch-interaktive Gantt-Diagramme in ihre Applikationen einbinden können.

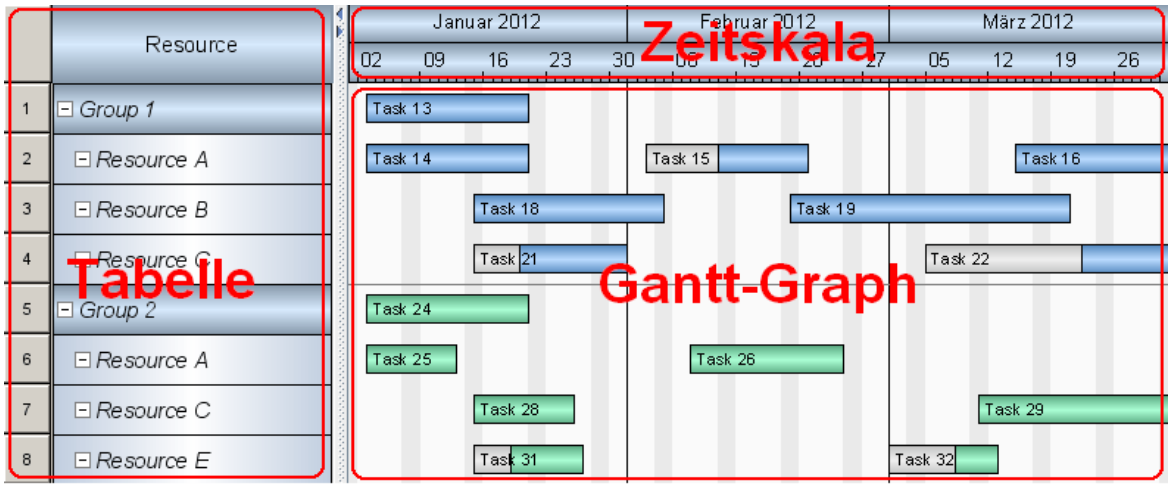
Gantt-Diagramme dienen dazu, zeitorientierte Prozesse wie Arbeitsvorgänge, Raum-, Maschinen- oder Bettenbelegung, Fuhrparkverwaltung, Landebahn- oder Hafenmanagement, Personaleinsatzplanung und andere Formen der Projekt- und Ressourcenplanung grafisch darzustellen. Gantt-Diagramme bilden Vorgänge über der Zeit ab.

Mit den Möglichkeiten eines über VARCHART JGantt erstellten Gantt-Diagramms können vor allem komplexe Prozesse übersichtlich gestaltet und wichtige Details auf einen Blick übermittelt werden. Das Gantt-Diagramm ist ein nicht mehr wegzudenkender Bestandteil von Software, die sich mit organisatorischen Abläufen befasst. Die interaktive Veränderbarkeit der grafischen Elemente durch den Benutzer macht das VARCHART Gantt-Diagramm zu einem wertvollen Planungsinstrument für Projektmanager.

Mit VARCHART JGantt können Java-Programmierer ihren Benutzern mit geringem Aufwand konfigurierbare und interaktive Gantt-Diagramme zur Verfügung stellen. Mit nur wenigen Mausklicks kann ein einfaches Gantt-Diagramm einer bestehenden Applikation hinzugefügt werden, das anschließend vielfach erweiterbar ist.

VARCHART JGantt wurde vollständig in der Programmiersprache Java geschrieben und ist daher auf allen gängigen Betriebssystemen lauffähig. Zur Entwurfszeit kann die JGantt-Komponente über Dialoge und zur Laufzeit über eine Programmierschnittstelle (API) konfiguriert werden, die mit dem „Eigenschaften-Editor“ adressiert werden kann.

1.2 Der Aufbau eines Gantt-Diagramms



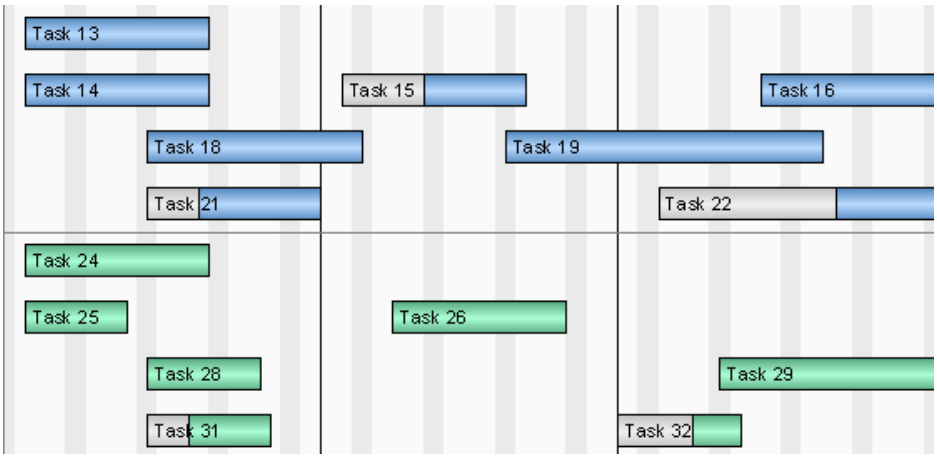
Ein Gantt-Diagramm stellt Vorgänge zeitbezogen dar. Es gibt drei grundlegende Elemente in einem Gantt-Diagramm:

- die Zeitskala
- den Gantt-Graphen
- die Tabelle

Die Zeitskala gibt die zeitliche Einteilung vor, z.B. ob die Vorgänge über Stunden, Tage, Monate u.a. abgebildet werden sollen.

Die Zeitskala kann mit VARCHART JGantt durch eine Reihe von grafischen Eigenschaften individuell gestaltet werden. Dazu gehören Zeiteinheiten, Farbmuster, Markierungsstriche, Schriftarten u.a.

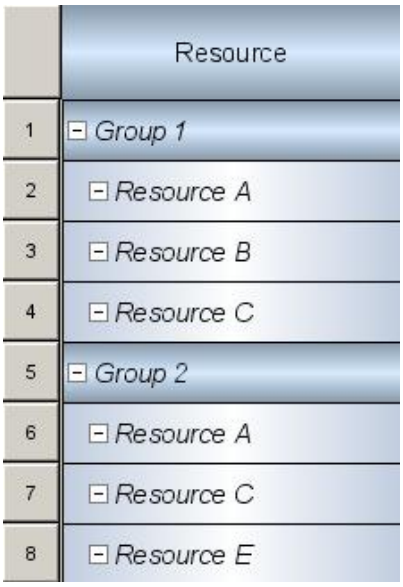
Im Gantt-Graphen werden die Vorgänge abgebildet. Sie heißen im VARCHART JGantt-Diagramm „Knoten“. Die Darstellung der Knoten basiert auf Daten, z.B. Anfangs- und Endzeitpunkt, der Ausführungsgrad und vieles andere mehr. Diese Details können mit VARCHART JGantt grafisch umgesetzt werden:



Besondere Zeitpunkte oder verschiedene Phasen eines Vorgangs können mit VARCHART JGantt unterschiedlich dargestellt werden. Auch Beschriftungen sind möglich. Die Zusammengehörigkeit von Vorgängen kann durch Gruppierung von Knoten visualisiert werden. Durch die vielfältige grafische Gestaltbarkeit sind die Details von Vorgängen auf einen Blick erkennbar.

Auch der Hintergrund des Diagrammbereichs kann gestaltet und strukturiert werden. Waagerechte Linien können Gruppierungen von Knoten anzeigen. Auch senkrechte Liniengitter sind möglich, die auf einen Blick das zeitliche Gefüge sichtbar machen, in das die Vorgänge eingebettet sind.

Unterhalb des Gantt-Graphen kann ein Histogramm Auslastungen aufsummieren und Kapazitäten anzeigen.



Die Tabelle zeigt einen ausgewählten Teil der Daten zu den Vorgängen an, wie z.B. Bezeichnungen von Gruppen, denen die Vorgänge angehören können, Anfangs- und Enddaten oder andere, die vom Entwickler oder Benutzer festgelegt werden können.

Mit den Möglichkeiten eines über VARCHART JGantt erstellten Gantt-Diagramms können komplexe Prozesse wie die Maschinenbelegung einer industriellen Produktionskette, die Phasen eines Großbauprojektes oder die Arbeitsgänge einer Filmproduktion übersichtlich gestaltet und wichtige Details auf einen Blick übermittelt werden. Die interaktive Veränderbarkeit der grafischen Elemente durch den Benutzer macht das VARCHART Gantt-Diagramm zu einem unentbehrlichen Werkzeug für Projektmanager.

1.3 Die VARCHART JGantt Handbücher

Es gibt für VARCHART JGantt zwei Handbücher: das Benutzerhandbuch und das Referenzhandbuch. Wenn Sie noch ganz am Anfang Ihrer Arbeit mit VARCHART JGantt stehen, sollten Sie zunächst das Benutzerhandbuch lesen. Damit erwerben Sie die Kenntnisse und Voraussetzungen, die Sie befähigen, Gantt-Diagramme mit VARCHART JGantt zu programmieren. Das Referenz-Handbuch dient dazu, bei der Programmierung Details nachzulesen und Ihre Grundkenntnisse um viele Details zu erweitern.

1.4 Die Installation von VARCHART JGantt

Um mit VARCHART JGantt Anwendungen zu entwickeln, sollten Sie über eine fertig eingerichtete Programmierumgebung für JAVA verfügen. Ihre JavaTM Plattform darf Stufe 2 Version 1.4 nicht unterschreiten. Die Hardwareanforderungen von VARCHART JGantt sind nicht höher als die einer durchschnittlichen JAVA-Programmierumgebung. Folgen Sie bei der Installation den Installationsanweisungen. Ziehen Sie anschließend die installierte Komponente auf ein leeres Fenster und spannen Sie sie über die gesamte Fenstergröße auf.

In einem Teilfenster sollte Sie die Tabelle der Eigenschaften sehen können.

Achten Sie bitte in der Eigenschaftentabelle darauf, dass die Orientierung (**Constraints**) des Gantt-Objektes auf die Mitte des Fensters ausgerichtet ist und die Einstellung **center** aufweist, ansonsten wird das Diagramm in einem Randbereich des Fensters abgebildet.

Nach der Installation können Sie mit der Programmierung eines Gantt-Diagramms beginnen.

1.5 Die Arbeitsweise mit VARCHART JGantt

1.5.1 Grundsätzliche Vorgehensweise

VARCHART JGantt ist ein Java-Bean, die in Komponenten unterteilt ist. Diese setzen sich aus spezifischen Klassen zusammen, die Methoden und Eigenschaft anbieten, mit denen Sie die sichtbaren und nicht sichtbaren Elemente des Gantt-Diagramms verwalten können. Eine zentrale Komponente ist die **JGantt**-Komponente, mit der Sie die meisten der für ein Gantt-Diagramm notwendigen Dinge einrichten können.

Eine weitere sehr wichtige Komponente ist die **AppData**-Komponente. Die AppData-Komponente hält zentral alle zur Visualisierung benötigten Anwendungsdaten. Diese können über die Programmierschnittstelle, alternativ über eine XML-Datei definiert und generiert werden, um die entsprechenden grafischen Darstellungen (z.B. "Knoten", Verbindungen) im Gantt-Graphen darzustellen.

Mit der **Kalender**-Komponente können Sie Sequenzen von Arbeitszeiten und Nicht-Arbeitszeiten im Gantt-Diagramm anlegen. Mit den Klassen der **Histogramm**-Komponente können Sie ein Histogramm generieren und die **Tabellen**-Komponente bietet spezifische Optionen für das Design und die Funktionalitäten der Tabelle. Mit der **Zeitberechnungs**komponente können Sie ein Projekt terminieren.

Im Design-Modus erstellen Sie ein Gantt-Diagramm mit den Elementen Zeitskala, Gantt-Graph, Tabelle und vielleicht auch einem Histogramm. Sie können hier das Diagramm entsprechend der Anforderungen Ihrer Applikation modellieren. Dafür können Sie entweder die Eigenschaften in der Eigenschaften-Pane Ihrer Programmierumgebung verwenden, das eine Teilmenge der Optionen anbietet, die in VARCHART JGantt vorhanden sind. Viele dieser Eigenschaften sind im Benutzerhandbuch erläutert. Alternativ, aber auch als Ergänzung, sollten Sie die Programmierschnittstelle (API) benutzen, die Ihnen die komplette Funktionalität von VARCHART JGantt zur Verfügung stellt. Ihre Klassen und Methoden sind im Referenzhandbuch erklärt.

Zur Laufzeit können Sie Knoten im Gantt-Diagramm interaktiv generieren oder verändern, womit Sie die entsprechenden Daten verändern. Jede Interaktion, d.h. jede Datenänderung wird in der **AppData**-Komponente abgelegt und wieder eingelesen.

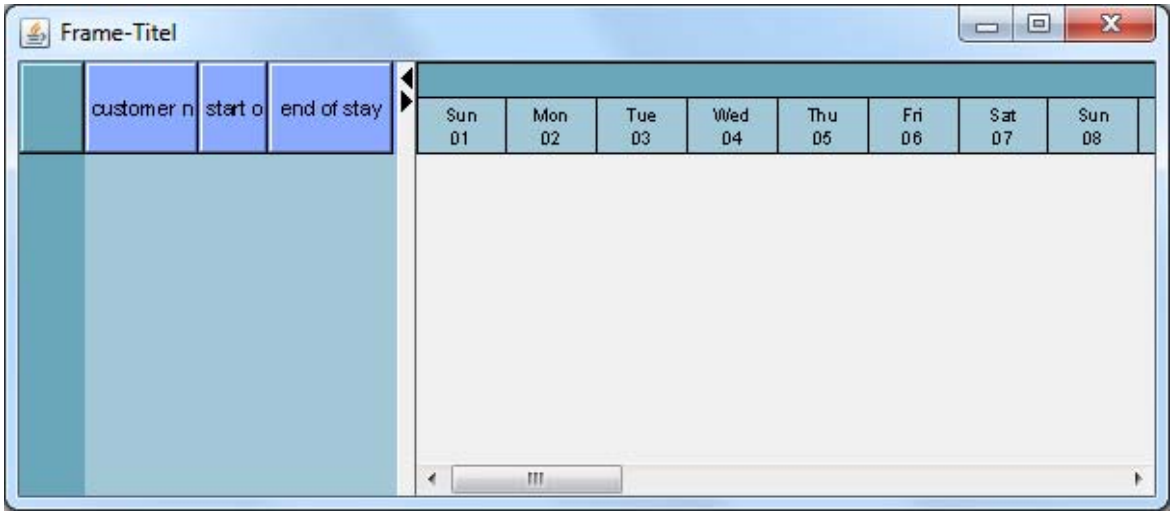
1.5.2 Nützliche Werkzeuge

Während der Entwicklungszeit oder auch wenn Sie Ihr Programm erstellt haben, ist es vor allem zum Ausprobieren verschiedener Eigenschaften ihres Diagramms sehr praktisch, den Eigenschaften-Editor zu verwenden. Damit können Sie schnell eine Eigenschaft in Ihrem Programm finden und ändern, um anschließend sofort das Ergebnis zu sehen. Sehr schnell lassen sich damit Elemente in den Diagrammen austauschen, Farben ändern oder Strukturen anders darstellen. Mit diesem Werkzeug können Sie z.B. im Dialog mit Ihrem Kunden sehr schnell Variationen erstellen und zeigen.

2 Die Sichtweise des Benutzers

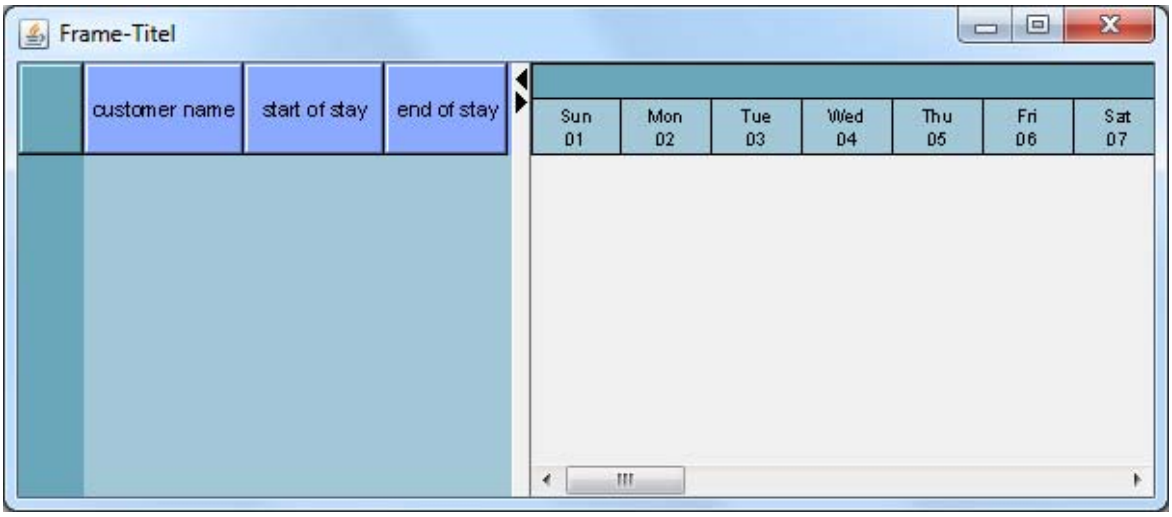
Bevor Sie damit beginnen, mit VARCHART JGantt ein Gantt-Diagramm zu programmieren, sollten Sie sich zunächst in die Situation Ihres Benutzers versetzen und ein bereits bestehendes Diagramm ausprobieren.

Öffnen Sie das mitgelieferte Beispielprojekt **beginner.jpx**, kompilieren Sie es und rufen Sie die entstandene Datei **beginner.jar** auf. Zunächst erhalten Sie ein leeres Gantt-Diagramm auf dem Bildschirm:



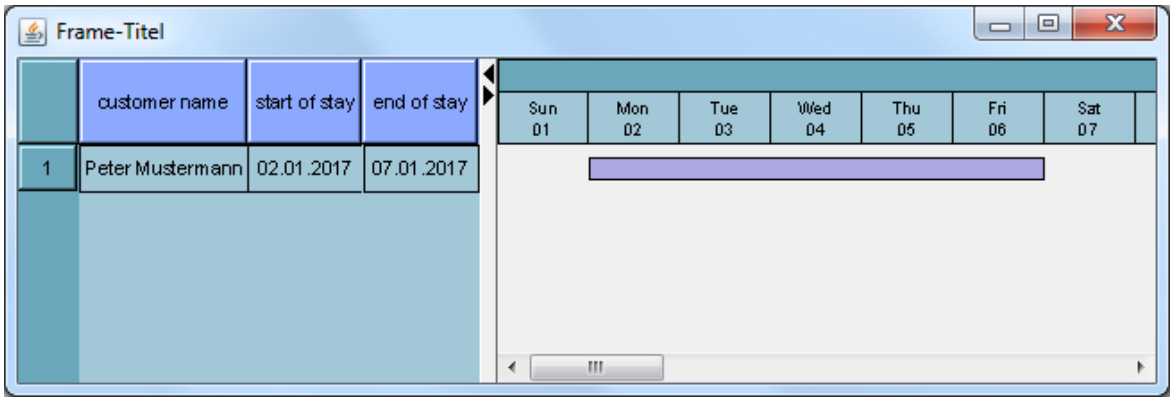
Es enthält links eine leere Tabelle, die nur eine Überschriftenzeile enthält und rechts die Zeitskala über einem leeren Gantt-Graphen.

Ziehen Sie als Erstes das Feld **name_of_customer** mit der Maus so breit, dass Sie den Namen vollständig lesen können. Anschließend ziehen Sie den Trennbalken zwischen Tabelle und Gantt-Graph so weit nach rechts, dass die Tabelle in ganzer Breite sichtbar ist. Die Tabelle zeigt nun drei Spalten: **name_of_customer**, **start**, und **end**. Diese Spalten sollen später die Namen von Kunden, den Beginn und das Ende ihres Aufenthaltes in einem Hotelzimmer anzeigen. Danach ziehen Sie das Fenster rechts noch ein wenig breiter, sodass sich die Zeitskala verbreitert:



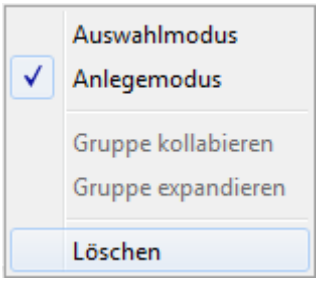
Legen Sie bitte nun einen Knoten an, indem Sie ihn mit der Maus in dem leeren Gantt-Graphen aufziehen. Sobald Sie die Maustaste loslassen, erscheint der folgende Dialog, in den Sie nun den Namen eines Kunden eintragen:

Nach der Eingabe erscheint der Knoten im Gantt-Diagramm:

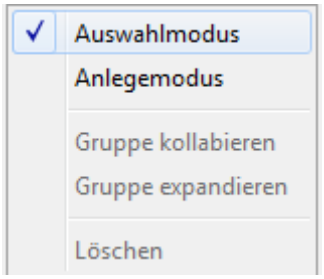


Sie sehen den Knoten rechts im Gantt-Graphen und gleichzeitig seine Daten links in der Tabelle.

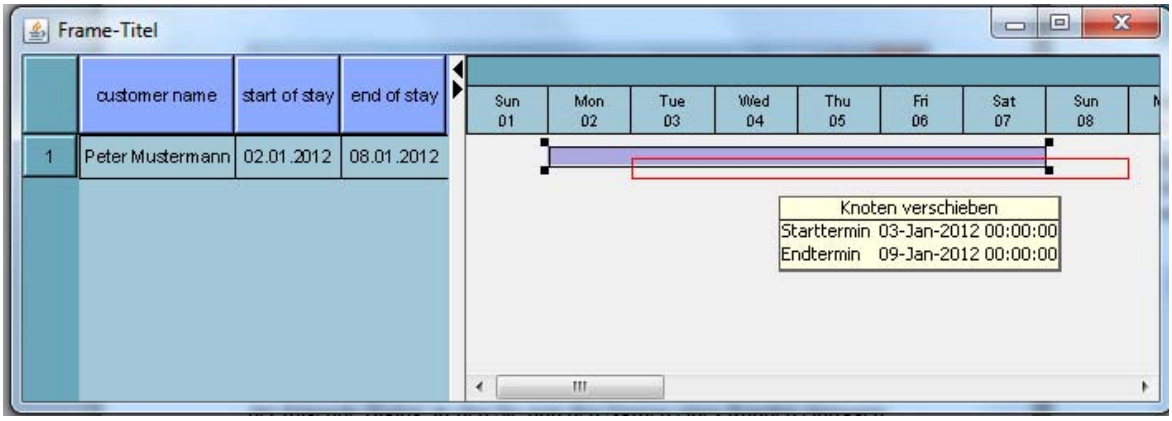
Legen Sie noch einen oder mehrere neue Knoten an. Markieren Sie diese einzeln durch Anklicken oder zu mehreren, indem Sie einen Rahmen darum herum ziehen. Drücken Sie nun in einem neutralen Bereich des Gantt-Graphen die rechte Maustaste und aktivieren Sie aus dem Kontext-Menü den Menüpunkt **Löschen**:



Bis hierher haben Sie sich im Anlegemodus für Knoten befunden. Drücken Sie nun in einem neutralen Bereich des Gantt-Graphen die rechte Maustaste und aktivieren Sie aus dem Kontext-Menü den **Auswahlmodus** für Knoten:



Markieren Sie nun den Knoten und ziehen Sie ihn mit der Maus ein wenig nach links oder rechts.



Sie werden dabei feststellen, dass ein rotes „Phantom“ erscheint, ein roter Rahmen, der - entsprechend Ihrer Mausbewegungen - die zukünftige Position des Knotens anzeigt.

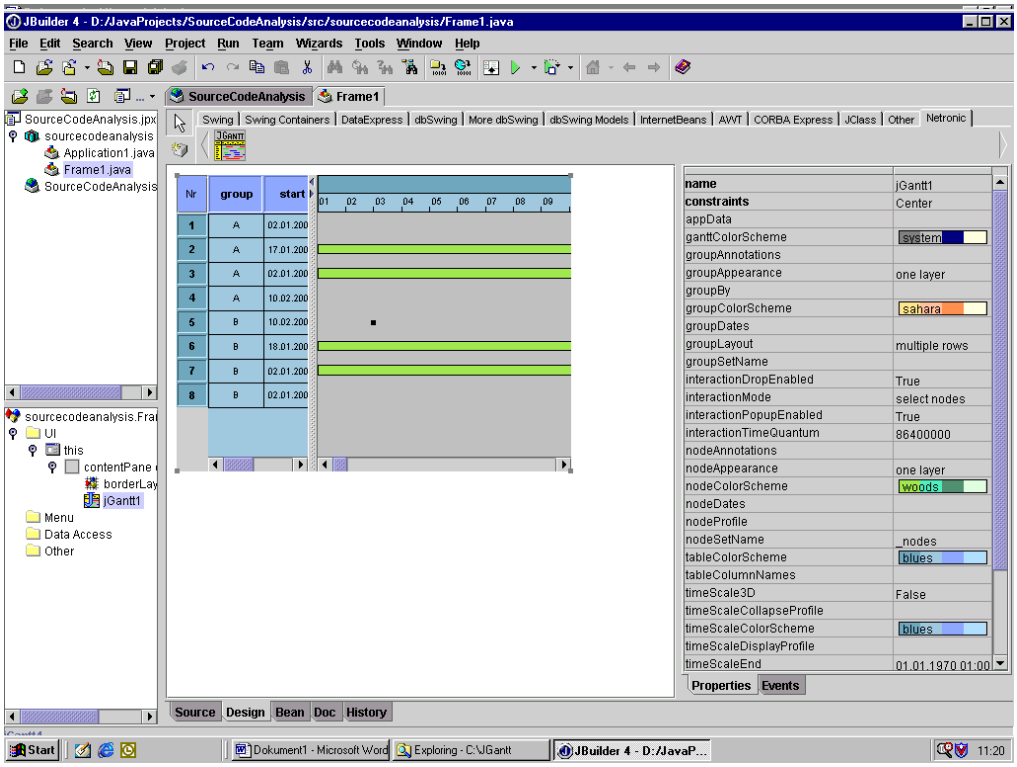
Zudem zeigt eine Infobox die neu eingestellten Daten an, so dass Sie eine sehr genaue Kontrolle bei der Positionierung haben. Die Daten werden auch gleichzeitig in der Tabelle aktualisiert, sobald Sie die Maustaste loslassen.

Sie können einen Knoten auch verlängern oder verkürzen. Dazu halten Sie die linke Maustaste über seinem Anfang oder Ende gedrückt und ziehen, ohne den Knoten vorher markiert zu haben.

Sie haben nun die wichtigsten grundlegenden Funktionen eines einfachen VARCHART JGantt-Diagramms ausprobiert: Sie haben interaktiv Vorgangsdaten angelegt, verändert und gelöscht. Außerdem haben Sie das Aussehen des Gantt-Diagramms interaktiv verändert. Die Änderungen der Vorgangsdaten wurden Ihnen sowohl über die Knoten im Gantt-Graphen als auch über die Zeilen in der Tabelle angezeigt. Sie können nun zur Programmierung eines solchen Diagramms fortschreiten.

3 Ein bestehendes Gantt-Diagramm verändern

Um Sie mit den Funktionsweisen von VARCHART JGantt vertraut zu machen, werden Sie nun zunächst das bestehende Programmbeispiel, das Sie vorhin ausprobiert haben, verändern. Kopieren Sie das Projekt unter anderem Namen auf Ihre Festplatte und öffnen Sie es im Design-Modus Ihrer Programmierumgebung. Sie erhalten das untenstehende Bild, das in allen gängigen Programmierumgebungen einen ähnlichen Aufbau zeigt:



Sie erhalten eine Abbildung des Gantt-Diagramms sowie eine interaktive Tabelle mit Eigenschaften des Gantt-Diagramms. Sie werden sich in den folgenden Kapiteln mit diesen Eigenschaften vertraut machen.

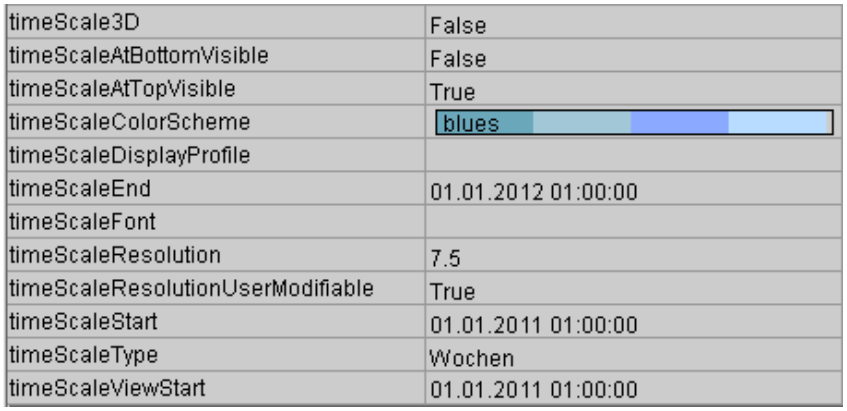
3.1 Veränderungen an der JGantt-Instanz

Wenn Sie wünschen, können Sie nun den Namen der Instanz Ihrer JGantt-Komponente ändern, z.B. um ihn in einen sinnreichen Zusammenhang mit

Ihrem Gesamtprojekt zu stellen. Standardmäßig heißt diese Instanz **jGantt1**. Klicken Sie auf den Namen und ändern Sie Ihn nach Ihren Wünschen.

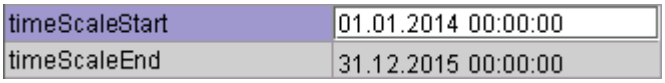
3.2 Veränderungen an der Zeitskala

Die Eigenschaften, mit denen Sie die Zeitskala verändern können, sind durch den Namensteil **timeScale** gekennzeichnet. Sie finden den Block aus **timeScale**-Eigenschaften im unteren Drittel der Eigenschaften-Tabelle:

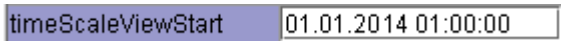


3.2.1 Anfangs- und Enddatum

Eine grundlegende Eigenschaft der Zeitskala ist ihre Ausdehnung. Sie wird durch den Anfangs- und Endtermin festgelegt und beschreibt gleichzeitig die Zeitspanne des gesamten Projektes. Klicken Sie auf das Anfangs- oder Enddatum der Zeitskala und editieren Sie die beiden Termine:



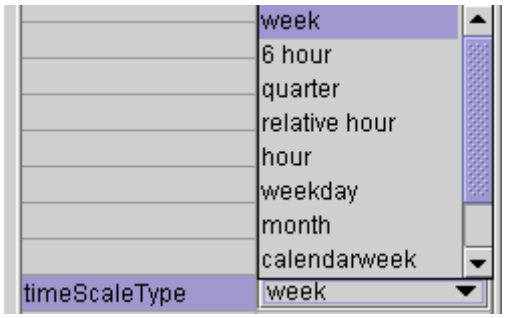
Außerdem können Sie festlegen, zu welchem Datum der Zeitskala gescrollt sein soll, wenn die Zeitskala erstmalig in einem Fenster erscheint:



Probieren Sie verschiedene Zeitpunkte aus und beobachten Sie die Veränderungen.

3.2.2 Zeitskalentyp

VARCHART JGantt bietet Ihnen verschiedene Zeitskalentypen an, die verschiedene Zeiteinheiten darstellen: Stunden, 6 Stunden, Tage, Wochen, Monate und Quartale. Neben den absoluten Werten können Sie auch relative Werte wie die Tagesziffern von 1-31, die Kalenderwochen von 1-52 oder die Tagesstunden von 1-24 anzeigen lassen. Die Sprache, in der diese Auswahlmöglichkeiten angezeigt werden, ist abhängig von Ihren Systemeinstellungen.



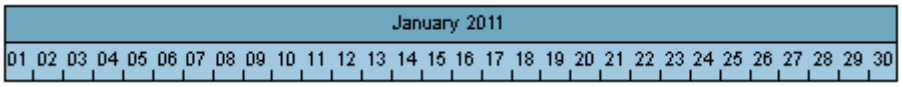
Probieren Sie die Parameter des **timeScaleType**-Feldes aus und beobachten Sie die Veränderungen an der Zeitskala.

3.2.3 Auflösung der Zeitskala

Das sinnvolle Einstellen der Zeitskalenauflösung ist abhängig von den Anfangs- und Enddaten sowie vom Zeitskalentyp. Die Auflösung wird ausgedrückt in Anzahl mm pro Einheit der Zeitskala. Wenn Sie z.B. hier die Zahl 10 eintragen und gleichzeitig eine Wochenskala wählen, wird eine Woche mit 10 mm Breite angezeigt. Wenn Sie eine Tagesskala wählen, wird ein Tag mit 10 mm Breite angezeigt.

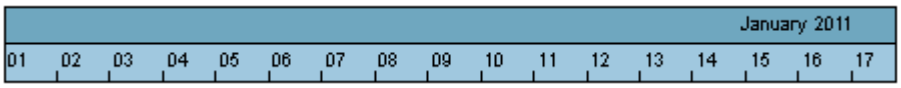
Zur Einstellung können Sie das Feld rechts von der Bezeichnung **timeScaleResolution** editieren:

Wenn Sie nun, als Beispiel, bei der Standardgröße des Fensters eine Zeitskala vom Typ **Tag** wählen und das Anfangsdatum auf den 1. und das Enddatum auf den 31. Januar setzen, erhalten sie bei der kleinstmöglichen Auflösung den gesamten Zeitstreifen vom Januar angezeigt:



Der kleinstmögliche Wert wird von VARCHART JGantt automatisch berechnet. Um ihn sichtbar zu machen, versuchen Sie, die Auflösung auf 1 zu setzen. Sie werden sehen, dass VARCHART JGantt diesen Wert nicht zulässt und einen eigenen berechnet. Dies erfolgt, damit die Zeitskala mit ihrer jeweiligen Länge immer mindestens so lang ist wie das Fenster.

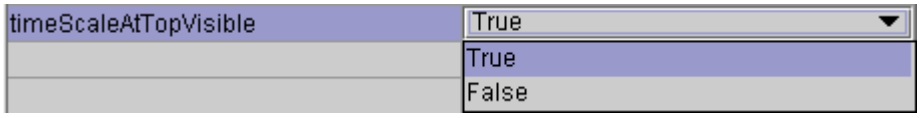
Setzen Sie die Auflösung nun höher, z.B. auf 5, erhalten Sie nur noch einen Teil Ihrer Zeitskala angezeigt:

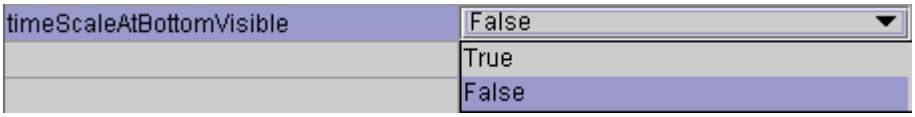


Probieren Sie verschiedene Zeitspannen und Auflösungen aus und beobachten Sie die Veränderungen an der Zeitskala.

3.2.4 Position der Zeitskala

Anzahl und Position der Zeitskala sind veränderbar. Sie können maximal zwei Zeitskalen, d.h. eine am oberen und eine am unteren Rand des Diagramms positionieren. Klicken Sie dazu in das Feld rechts neben **timeScaleAtTopVisible** und **timeScaleAtBottomVisible**. Die Combo-Box bietet Ihnen die Möglichkeit, die entsprechende Zeitskala auf sichtbar oder unsichtbar zu schalten:

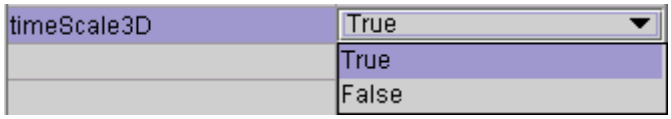




Probieren Sie verschiedene Möglichkeiten aus und beobachten Sie die entsprechenden Veränderungen.

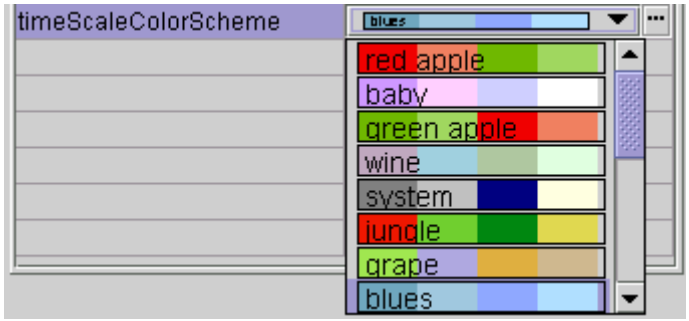
3.2.5 3D-Effekt

Weiterhin können Sie festlegen, ob die Zeitskala zwei- oder dreidimensional dargestellt werden soll. Sie können dazu die dreidimensionale Ansicht auf True oder False setzen. Beobachten Sie die Veränderungen im Diagramm.



3.2.6 Farbgebung

Eine weitere Gestaltungsmöglichkeit ist die Zuweisung eines Farbschemas, mit dem Sie die Zeitskala hinterlegen wollen. Klicken Sie dazu bitte auf das aktive Farbschema rechts neben dem Feld **timeScaleColorScheme**. Das sich öffnende Listenfeld bietet Ihnen weitere Farbschemata an. Wählen Sie eins aus und beobachten Sie die entsprechenden Farbänderungen an der Zeitskala.



Ein Farbschema besteht aus neun Farben, von denen acht zu vier Paaren zusammengefasst sind, die sich gegenseitig in der Form von Hintergrund- und Schriftfarbe ergänzen:

2 der **mainColor** (Hintergrund zur textColor)

3 der **textColor** (Schriftfarbe zur mainColor)

4 der **shadedColor** (Hintergrund zur shadedTextColor)

5 der **shadedTextColor** (Schriftfarbe zur shadedColor)

6 der **alternateColor** (Hintergrund zur alternateTextColor)

7 der **alternateTextColor** (Schriftfarbe zur alternateColor)

8 der **shadedAlternateColor** (Hintergrund zur **shadedAlternateTextColor**)

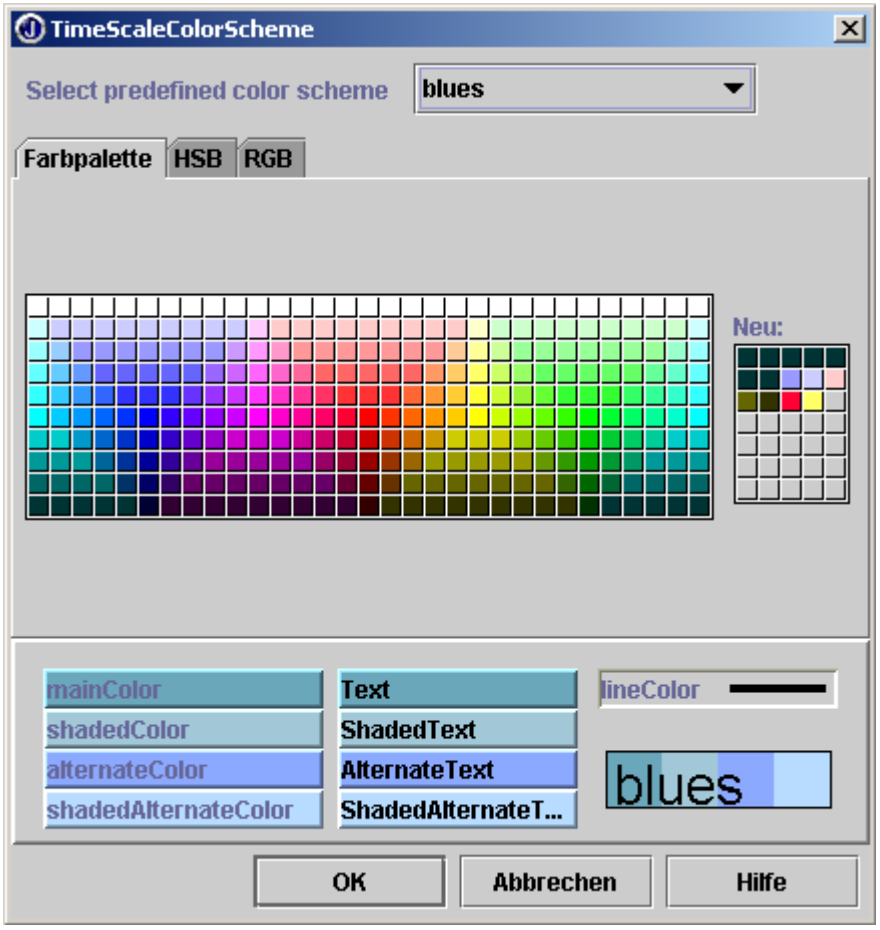
9 der **shadedAlternateTextColor** (Schriftfarbe zur **shadedAlternateColor**)

10 der **lineColor**

Je nach Objekt, für das ein Farbschema bestimmt ist, werden diese fünf Farbgruppen unterschiedlich verwendet. In unserem Beispiel der Zeitskala werden die **mainColor** und die **shadedColor** für die beiden Zeitstreifen verwendet. Mit den jeweiligen Schriftfarben können Sie die Schrift der Zeitstreifen beeinflussen.

Die **alternateColor** und die **shadedAlternateColor** dienen der farblichen Hinterlegung eines Kalenderstreifens, der z.B. Arbeitstage und Wochenende als regelmäßiges Muster farblich voneinander absetzt (die Anlage von Kalenderstreifen lernen Sie in einem späteren Kapitel). Da Kalenderstreifen nicht beschriftbar sind, kommen die entsprechenden Schriftfarben bei der Zeitskala nicht zur Anwendung, bei anderen Objekten, auf die ein Farbschema angewendet wird, jedoch durchaus. Die **lineColor** wird für die Trenn- und Rahmenlinien der Zeitskala verwendet.

Falls Sie ein bestehendes Farbschema ändern wollen, wählen Sie es aus und klicken anschließend auf die Schaltfläche mit den drei Punkten rechts daneben. Es öffnet sich der Dialog **TimeScaleColorScheme**, mit dem Sie jede Farbe eines Farbschemas bearbeiten können:

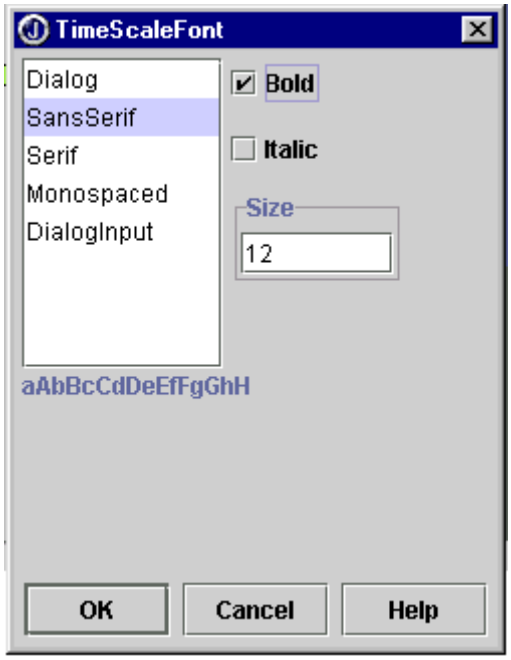


Klicken Sie dazu eine der Farben im Farbschema an, z.B. die **mainColor** und wählen Sie anschließend aus dem Farbraster eine neue Farbe aus, mit der Sie **mainColor** ersetzen wollen. Alternativ können Sie die Farbwerte auch über das HSB-Schema oder RGB-Werte stufenlos einstellen. Sobald Sie eine der vier vorgegebenen Farben verändern, wechselt der Name des Farbschemas zu CUSTOM. Ein Custom-Schema ist nur gültig für die aktuelle Anwendung und ist mit ihr speicherbar. Sobald Sie das Custom-Schema wiederum gegen ein anderes Schema austauschen, wird das Custom-Schema verworfen.

3.2.7 Schriftarten

Schriftarten der Zeitskala können Sie ändern, indem Sie die Schaltfläche mit den drei Punkten rechts neben dem Feld **timeScaleFont** drücken:

Sie gelangen zu dem Dialog **TimeScaleFont**, in dem Sie zwischen fünf Schrifttypen wählen und Schriftgrad und Größe einstellen können.



Probieren Sie einmal verschiedene Schrifttypen aus und beobachten Sie die entsprechenden Veränderungen an der Zeitskala.

Erweiterte Funktionen der Zeitskala werden Sie in einem späteren Kapitel noch kennen lernen.

3.3 Veränderungen an Knoten

Die Gestaltung von Knoten kann in VARCHART JGantt über zwei verschiedene Wege erfolgen: Eine völlig freie Gestaltung können Sie über die Programmierschnittstelle vornehmen, oder Sie können über die Eigenschaftentabelle Knoten aus vordefinierten Elementen zusammensetzen. Erstere Option ist Kapitel 5.3.4 beschrieben, die letztere stellen wir hier vor.

Die Eigenschaften, mit denen Sie Knoten verändern können, sind in der Eigenschaftentabelle durch den Namensteil **node** gekennzeichnet. Sie finden den Block mit den Knoteneigenschaften etwa in der Mitte der Tabelle:

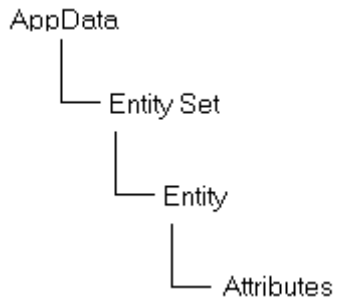
nodeAnnotations	
nodeColorScheme	wine
nodeDateFormat	
nodeDates	start<NL>end<NL><NL><NL
nodeDesign	Rect
nodeFont	"Dialog", 1, 7
nodePhantomPositionLine	no line
nodeProfile	
nodeProfileBy	
nodeSetName	Nodes
nodesSortedBy	
nodeSymbols	--- none ---, --- none ---
nodeZeroLengthSymbol	--- none ---
nodeZeroLengthVisible	False

Mit Ausnahme der Eigenschaften **NodeProfile** und **NodeProfileBy**, die in einem späteren Kapitel behandelt werden, stellen wir Ihnen in den folgenden Unterkapiteln die aufgelisteten Eigenschaften vor.

3.3.1 Anlage und Auswahl des Entitäten-Sets

Jede Darstellung eines Knotens basiert auf Daten, die in seiner Entität abgelegt sind. Eine Entität ist Bestandteil einer Datenstruktur, die hierarchisch aufgebaut ist. Diese Hierarchie ist dem Aufbau einer Datenbank vergleichbar. Zentrales Element ist der Datensatz, der einerseits aus Feldern besteht und andererseits in einer Tabelle gespeichert ist. Analog zum Datensatz ist das zentrale Element in einer Objekthierarchie die "Entität". Eine Entität besteht aus Attributen, vergleichbar den Datensatzfeldern. Entitäten sind zu einem "Entitäten-Set" zusammengefasst, ähnlich einer Tabelle, und von allen Enti-

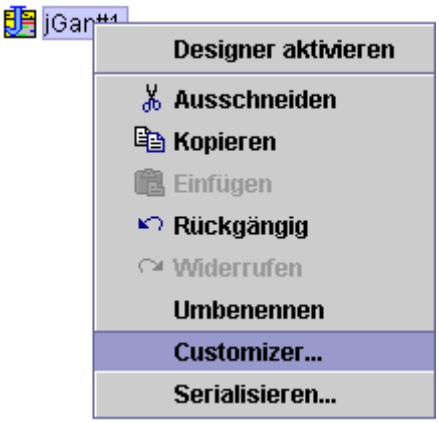
täten-Sets gibt es eine Liste oder Sammlung aller Applikationsdaten (application data, abgekürzt **AppData**), was einer Datenbank entspricht.



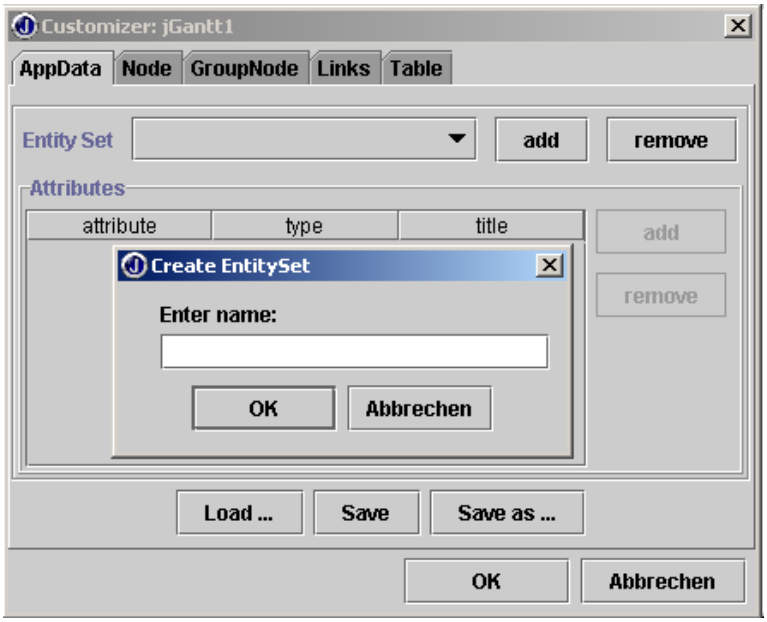
Der Unterschied zwischen den beiden analogen Strukturelementen ist der, dass in den Entitäten Objekte gespeichert werden, während in einem Datensatz einfache Daten abgelegt werden.

Die Einrichtung der **appData** mit ihren Entitäten-Sets, den Entitäten und ihren Attributen erfolgt während der Designzeit. Sie können die Erstellung der Datenstruktur besonders komfortabel mit Hilfe des Customizers vornehmen.

Dazu klicken Sie im Strukturfenster Ihrer Programmierumgebung (Design-Modus) mit der rechten Maustaste auf das JGantt-Element und erhalten das folgende Kontext-Menü:

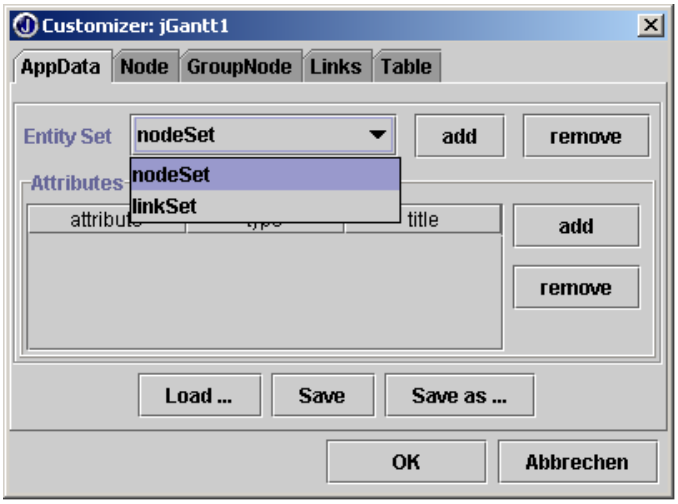


Hier können Sie den Customizer anklicken. Sie erhalten einen Dialog, der Ihnen verschiedene Karteikartenreiter zur Auswahl anbietet und zunächst die Einrichtung der **AppData** anbietet. Durch Klicken auf die Schaltfläche **add** können Sie nun Entitäten-Sets einrichten. Legen Sie z.B. ein Entitäten-Set für Knoten an mit dem Namen nodeSet:

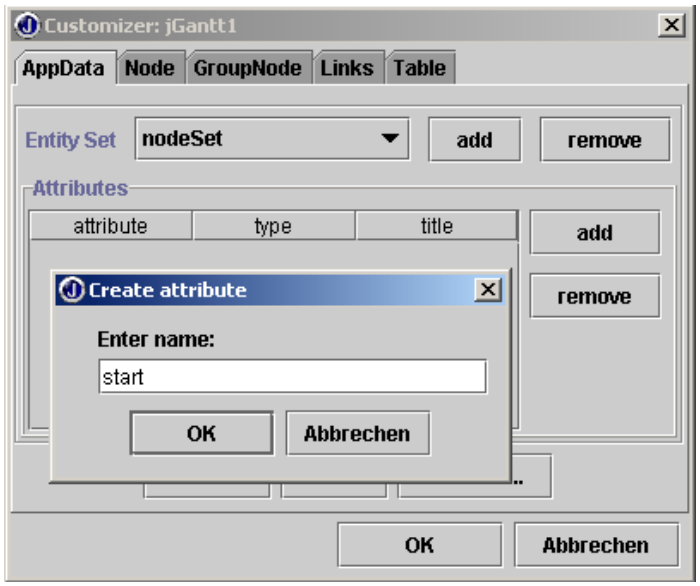


Sie können die angelegten Entitäten-Sets einer **AppData** über die Schaltfläche **Save** als XML-Datei speichern, mit **Save as** unter anderem Namen speichern oder andere XML-Dateien über die Schaltfläche Load laden. Es sei hier noch einmal betont, dass es hier nur um die Einrichtung der Datenstruktur, nicht um die Anlage von Daten geht. Aus diesem Grund können Sie nur solche XML-Dateien laden und einlesen, die ausschließlich eine Datenstruktur enthalten (aber keine Daten).

Sofern Sie nur ein einziges Entitäten-Set angelegt haben, wird dieses automatisch ausgewählt. Sind dem Customizer mehrere bekannt, nehmen Sie die Auswahl manuell vor:



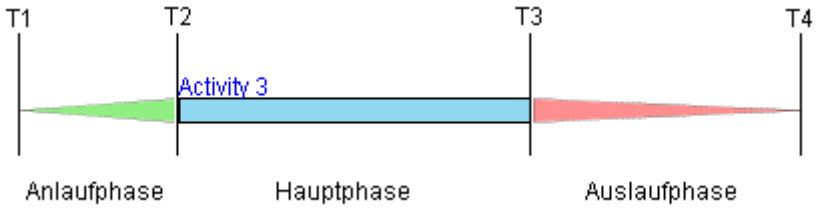
Nachdem Sie ein Entitäten-Set angelegt und ausgewählt haben, werden die gegrauten Flächen add und remove im Fenster Attributes aktiv, mit denen Sie Attribute der Entitäten anlegen und löschen können:



Doch zunächst noch einige nützliche Informationen zu den Daten, mit denen Sie ihre Knoten-Entität bestücken.

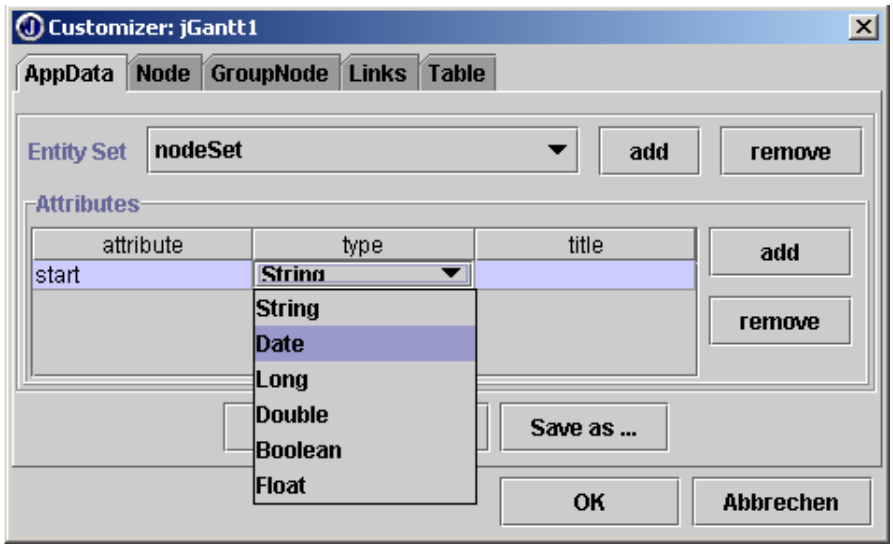
3.3.2 Knotentermine

Ein Knoten besteht meist aus vier Terminen: T1, T2, T3 und T4. Diese Termine können in verschiedenen Anwendungen verschiedene Bedeutungen haben: z.B. den frühest- und spätestmöglichen Anfang sowie das frühest- und spätestmögliche Ende des Vorgangs. Oder sie können den Beginn einer Anlaufphase, den Beginn der Hauptphase, das Ende der Hauptphase und das Ende der Auslaufphase darstellen.

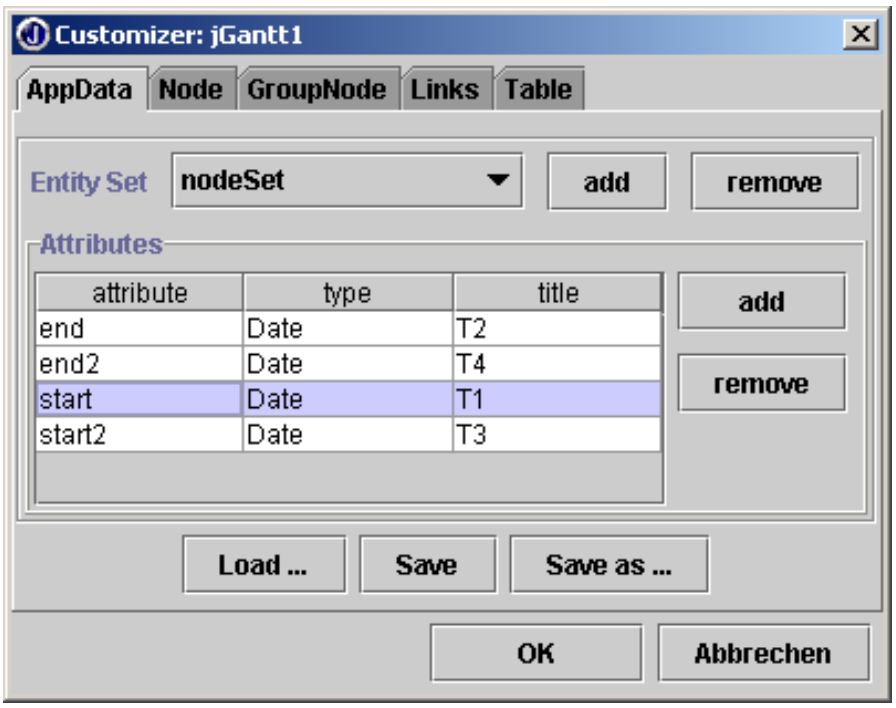


Nicht alle vier Termine in einem Knoten müssen genutzt werden – manche Knoten kommen bei ihrer Darstellung auch mit einem oder zwei Terminen aus. Es macht aber in den meisten Fällen Sinn, als Datengrundgerüst für einen Knoten vier Termine einzurichten.

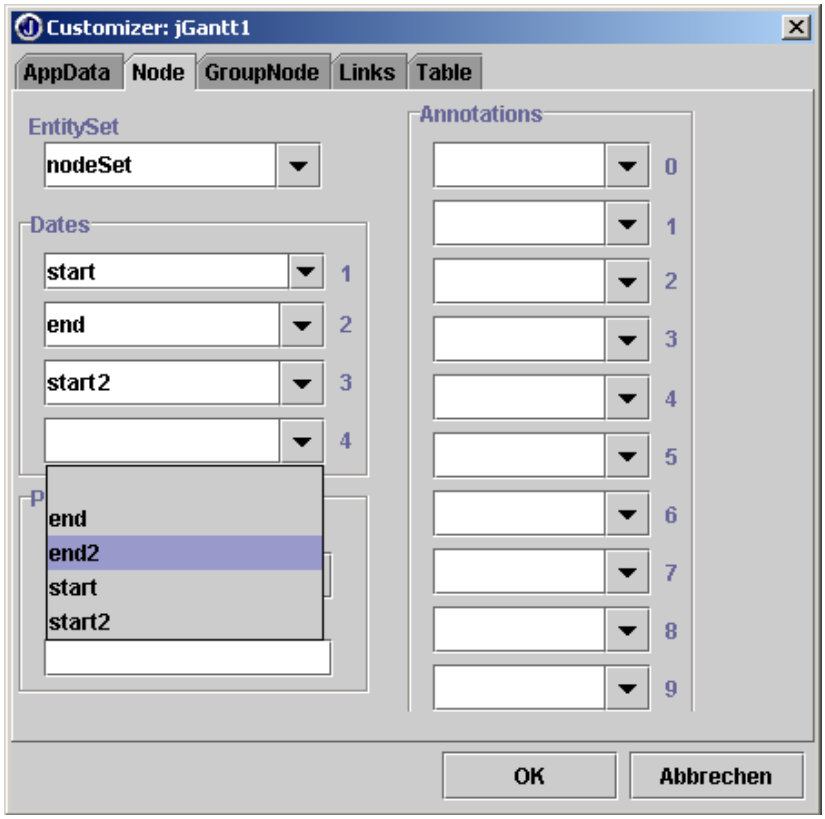
Die Anlage von Datenstrukturen nehmen Sie mit Hilfe des Customizers vor und richten hier nun vier Termine ein. Dazu wählen Sie nach dem Aufruf die Karteikarte **AppData** als Entitäten-Set nodeSet aus. Dann klicken Sie im Feld Attributes auf die Schaltfläche **add** und legen ein Attribut an, das Sie **start** nennen. Standardmäßig erhalten Attribute den Datentyp String, den Sie anschließend editieren können:



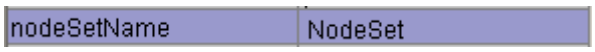
Sie können dem Attribut auch einen Titel geben, der später in der Tabelle als Spaltenüberschrift angezeigt werden kann.



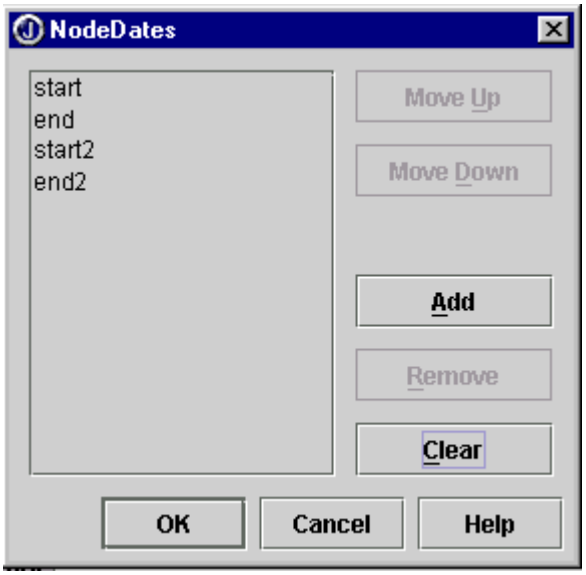
Wichtig nach der Anlage der Datenstruktur ist nun, dem Gantt-Diagramm bekannt zu geben, dass das Entitäten-Set **nodeSet** die Daten zur Darstellung der Knoten liefert, und dass die Attribute **start**, **end**, **start2** und **end2** die vier Termine der Knoten enthalten. Dazu wählen Sie die Karteikarte Node aus, setzen im Feld EntitySet das Entitäten-Set nodeSet und setzen die vier Termine:



Die eingerichteten und bekannt gegebenen Datenstrukturen werden nun auch in der Eigenschaftentabelle im Design-Mode Ihrer Programmierumgebung sichtbar und Sie können diese hier für Ihr Gantt-Diagramm verändern, falls Bedarf ist:



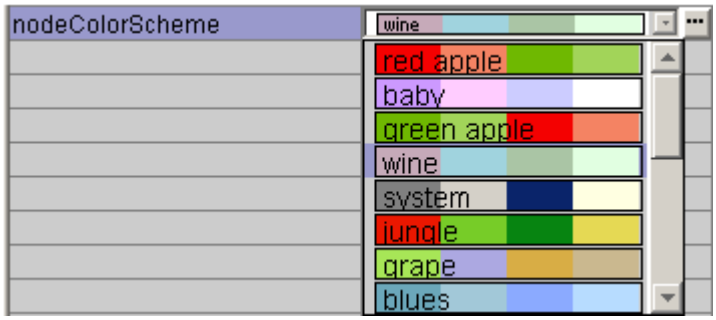
Wenn Sie auf die Schaltfläche mit den drei Punkten drücken, erscheint der folgende Dialog, mit dessen Hilfe Sie die Festlegungen editieren können. Sie können die Reihenfolge der Einträge mit den Schaltfläche **Move up** und **Move down** verschieben und damit die Namen vertauschen. Mit **Remove** entfernen Sie einzelne, mit **Clear** löschen Sie alle Attributnamen.



Auf diese Weise können Sie die Datenattribute einer Entität auch in der Eigenschaftentabelle bearbeiten.

3.3.3 Die Farbgebung des Knotens

Auch den Knoten können Sie, wie anderen Elementen des Gantt-Diagramms, ein Farbschema zuweisen. Sie haben dies bereits bei der Zeitskala ausprobiert und sollten sich auch beim Knoten ansehen, wie die vier Farben verwendet werden.



Wie sie im Detail für die unterschiedlichen Knotendesigns zur Verwendung kommen, finden Sie in der Tabelle des folgenden Kapitels **Das Design des Knotens**.

Die Farbgebung der Knoten über ein solches Farbschema ist statisch, d.h. alle Knoten weisen dieselbe Farbe auf. Alternativ können Sie die statischen Farben durch dynamische Farben ersetzen und damit die Farbe einzelner Knoten

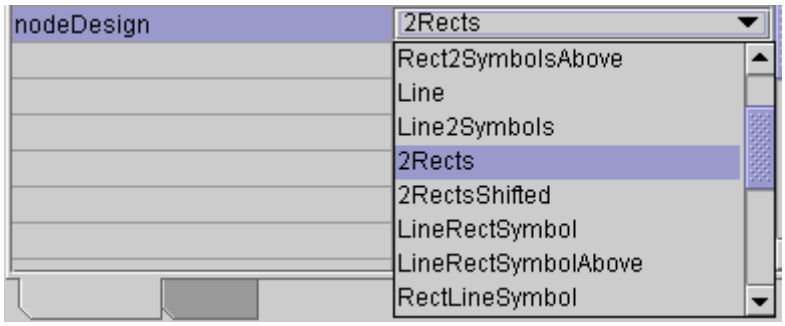
datengesteuert anders gestalten. Die dynamische Farbgebung wird jedoch erst in einem späteren Kapitel erläutert.

Die Änderung der Farben eines bestehenden Farbschema nehmen Sie in gleicher Weise vor wie bei der Zeitskala.

3.3.4 Das Design des Knotens

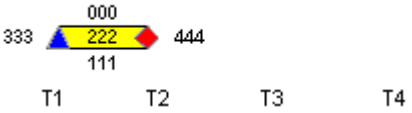
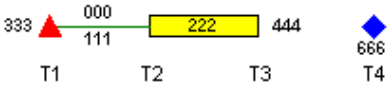
Wie Sie bereits aus dem Kapitel **Knotentermine** wissen, besitzt jeder Knoten in der Regel vier Termine: T1, T2, T3 und T4. Die Zeitspannen zwischen diesen Terminen können Sie durch verschiedene Balken- oder Linienformen darstellen. Weiterhin können Sie dem Design verschiedene Symbole zuweisen und ober- und unterhalb der Balken und Linien Beschriftungen positionieren.

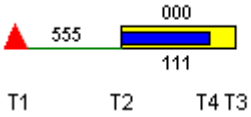
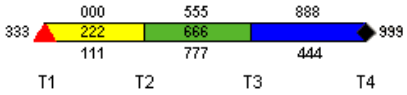
Mit der Eigenschaft **nodeDesign** legen Sie das Design der Knoten fest. Wenn Sie das Feld rechts daneben anklicken, erhalten Sie in der Combo-Box eine Liste mit einer Auswahl an verschiedenen Knotenformen.

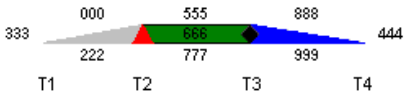


Wenn Sie diese Knoten-Designs ausprobieren, weisen Sie einem Knoten zusätzlich zwei Knotensymbole (nächstes Kapitel) und 10 Beschriftungen (übernächstes Kapitel) zu. Sie werden erst zusammen mit den Knotensymbolen und den Knotenbeschriftungen zu den Ergebnissen kommen, die in der folgenden Tabelle abgebildet sind.

Die Tabelle listet einige der Knoten-Designs auf und erläutert ihre Merkmale. Die komplette Darstellung aller möglichen Knoten-Designs finden Sie im Referenzhandbuch. Zur Erläuterung des Beschriftungsindex lesen Sie bitte das Kapitel Knotenbeschriftungen“.

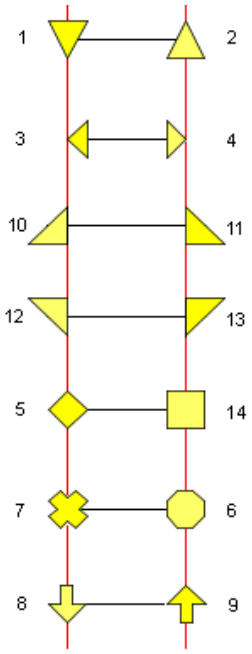
Bezeich-nung	Merkmale	Darstellung
Symbol	Ein Symbol, falls vorhanden, wird in der Farbe shadedColor auf T1 dargestellt. Eine Beschriftung durch den Index 4 rechts am Symbol ist möglich.	
Rect	Nur der Zwischenraum des Terminpaars T1-T2 wird durch einen breiten Balken der Farbe shadedColor dargestellt. Diesem kann oben und unten eine Beschriftung über die Indices 0 bzw. 1 zugewiesen werden, links und rechts über die Indices 3 bzw. 4 und mittig über den Index 2. Vorhandene Symbole werden T1 in der Farbe mainColor und T2 in der Farbe alternateColor zugewiesen. Der Zwischenraum des zweiten Datenpaars T2-T3-T4 bleibt leer.	
LineRect-Symbol	Der Zwischenraum zwischen T1-T2 wird durch eine Linie der Farbe shadedAlternateColor dargestellt, die oben und unten mit Beschriftungen der Indices 0 bzw. 1 versehen werden kann. T1 wird mit einem vorhandenen Symbol in der Farbe alternateColor belegt und über den Index 3 links mit einer Beschriftung versehen. Der Zwischenraum zwischen T2-T3 wird mit einem breiten Balken der Farbe shadedColor dargestellt. Er kann innerhalb über den Index 2 und rechts über den Index 4 beschriftet werden. T4 wird mit einem vorhandenen	

	Symbol der Farbe mainColor besetzt und kann über den Index 6 unterhalb beschriftet werden.	
LineCompletionRect	Der Zwischenraum T1-T2 wird durch eine tiefgestellte Linie der Farbe shadedAlternateColor dargestellt und kann über den Index 5 oben beschriftet werden. T1 wird mit einem Symbol der Farbe alternateColor besetzt. Der Zwischenraum T2-T3 wird mit einem breiten Balken ausgefüllt der Farbe shadedColor und kann oben und unten über die Indices 0 bzw. 1 beschriftet werden. Der Zwischenraum T2-T4 wird mit einem schmalen Balken der Farbe mainColor ausgefüllt, der als „Fortschrittsbalken“ fungieren kann.	
3Rects	Der Zwischenraum zwischen T1-T2 wird durch einen Balken der Farbe shadedColor dargestellt und kann mittig, oben, unten und links über die Indices 2, 0 bzw. 1 beschriftet werden. Der Zwischenraum zwischen T2-T3 wird mit einem Balken der Farbe shadedAlternateColor ausgefüllt und kann mittig, oben und unten über die Indices 6, 7 bzw. 8 beschriftet werden. Der Zwischenraum zwischen T3-T4 wird mit einem Balken der Farbe mainColor ausgefüllt und kann oben, unten und rechts über die Indices 8, 4 bzw. 9 beschriftet werden. T1 ist mit einem vorhandenen Sym-	

	bol der Farbe alternateColor besetzt, das linksseitig über den Index 3 beschriftet werden kann. T4 ist mit einem vorhandenen Symbol der Farbe lineColor besetzt, das rechtsseitig über den Index 9 beschriftet werden kann.	
Ramps	Der Zwischenraum zwischen T1-T2 wird durch eine aufsteigende Rampe der Farbe shadedColor dargestellt, die oben, unten und links über die Inidces 0, 2 bzw. 3 mit einer Beschriftung versehen werden kann. Der Bereich zwischen T2 und T3 wird durch einen Balken der Farbe shadedAlternateColor dargestellt, der oben, mittig und unten über die Indices 5, 6 bzw. 7 beschriftet werden kann. Der Zwischenraum T3-T4 wird durch eine absteigende Rampe der Farbe mainColor dargestellt, die über die Indices 8, 9 bzw. 4 beschriftet werden kann. T2 ist mit einem vorhandenen Symbol der Farbe alternateColor, T3 ist mit einem vorhandenen Symbol der Farbe lineColor besetzt.	
summary	Der Zwischenraum zwischen T1-T2 wird durch einen Summenbalken mit hakenförmigen Enden in der Farbe mainColor dargestellt.	

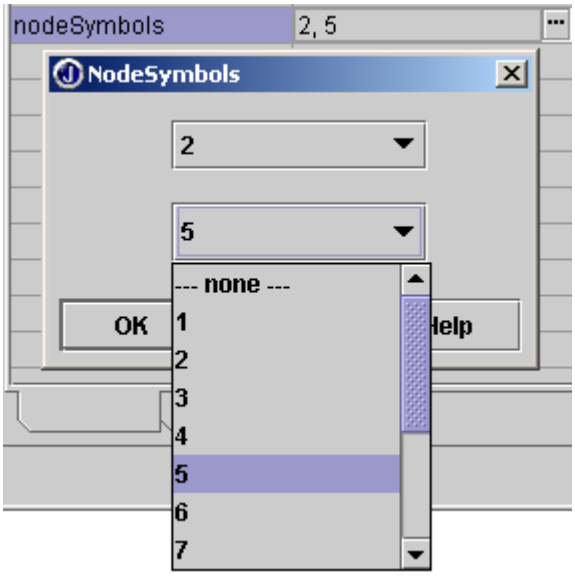
3.3.5 Knotensymbole

Ein Knotensymbol dient dazu, einen Termin innerhalb eines Knotens optisch in besonderer Weise hervorzuheben. Über die Eigenschaften-Pane können Sie einem Knoten maximal zwei Knotensymbole zuweisen (mehr sind möglich, wenn Sie Knoten über die Programmierschnittstelle erstellen – ein Kapitel darüber folgt später). Folgende Symbole stehen Ihnen zur Verfügung:



Anhand der senkrechten Linie können Sie die Position des Symbols relativ zu dem Datum erkennen, dem es zugewiesen ist. Einige Symbole sind mittig, andere randständig auf das Datum positioniert.

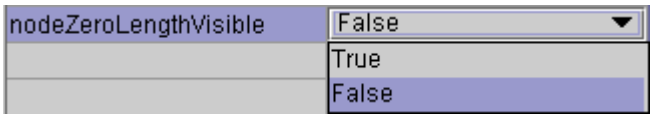
Die beiden zugewiesenen Symbole werden im Feld der Eigenschaft **nodeSymbols** angezeigt. Sie können die Symbole zuweisen oder zugewiesene verändern, indem Sie das Feld mit den drei Punkten anklicken. Damit gelangen Sie zu dem Dialog **NodeSymbols**, der Ihnen zwei Felder zum Auswählen der Symbol-Nummern anbietet.



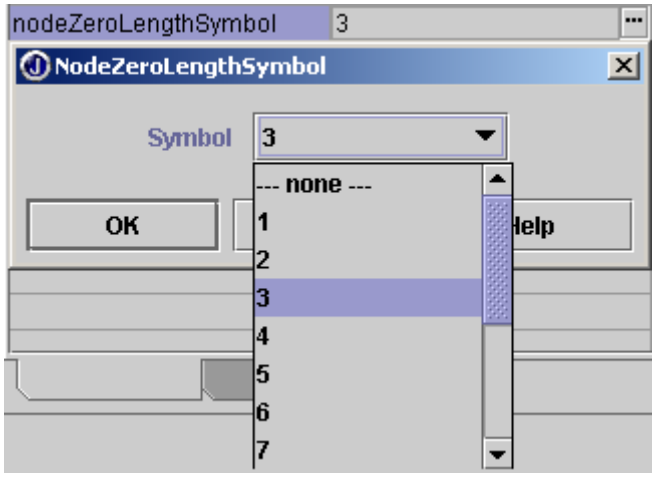
Um die Veränderungen, die Sie vornehmen, an Ihrem Beispiel sehen zu können, ist es notwendig, dass Sie ein Knoten-Design verwenden, das beide Symbole darstellt, z.B. das Knoten-Design **RectLineSymbol**.

3.3.6 Knoten ohne Ausdehnung

Knoten ohne Ausdehnung, d.h. Knoten, bei denen das Anfangs- und Enddatum identisch sind, können Sie wahlweise überhaupt nicht, als senkrechten kleinen Strich oder als Symbol darstellen. Die beiden ersteren Optionen stellen Sie über die Eigenschaft **NodeZeroLengthVisible** ein:



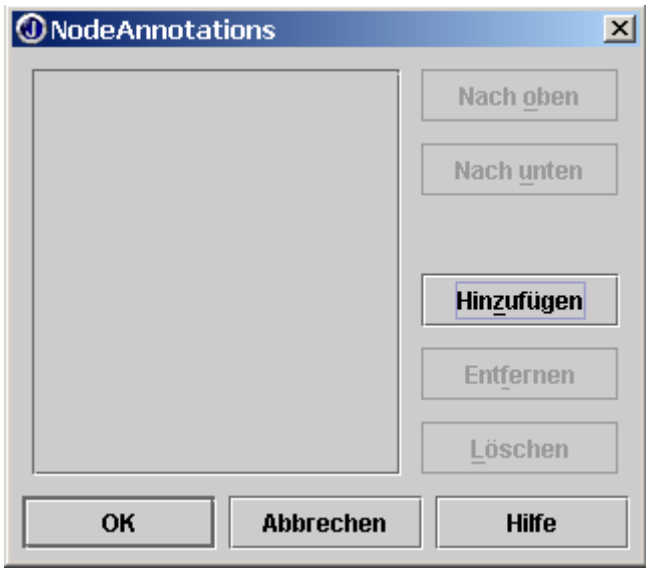
Mit **False** stellen Sie solche Knoten auf unsichtbar, mit **True** auf sichtbar. Wenn Sie die Knoten auf sichtbar stellen, haben Sie damit die Voraussetzung geschaffen, ihnen ein Symbol zuzuweisen. Dazu klicken Sie auf die drei Punkte im Feld ganz rechts bei der Eigenschaft **nodeZeroLengthSymbol**. Sie gelangen zu der Dialogbox **NodeZeroLengthSymbol**, in der Sie über eine Auswahlbox ein Symbol zuweisen können. Im Beispiel wurde das Symbol Nr. 3 gewählt:



3.3.7 Die Knotenbeschriftung

Mit der Eigenschaft **nodeAnnotations** können Sie die Beschriftung der Knoten einrichten oder ändern.

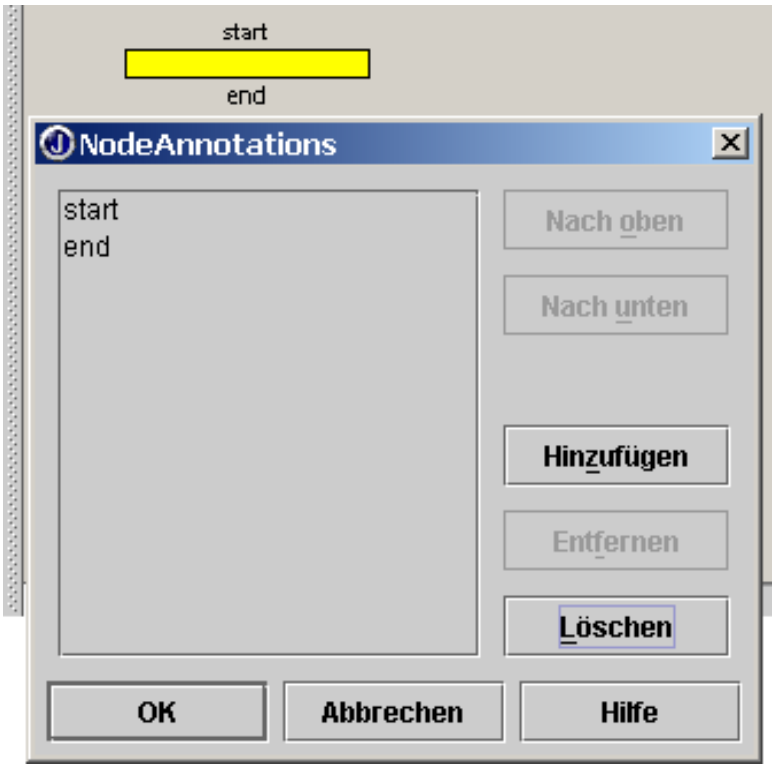
Klicken Sie dazu bitte auf die Schaltfläche mit den drei Punkten. Sie gelangen zu dem Dialog **NodeAnnotations**.



Knotenbeschriftungen werden aus Daten in Attributen der Knoten-Entität generiert. Für diese Generierung drücken Sie nun bitte die Schaltfläche Hinzufügen. Eine neue Beschriftung mit dem Namen **Eintrag 1** entsteht. Diese

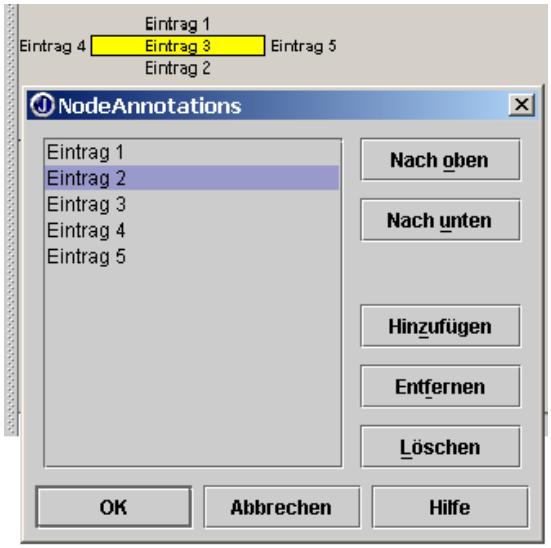
können Sie mit einem Doppelklick editieren. Geben Sie bitte dort den Namen **start** ein. Dadurch weisen Sie der Knotenbeschriftung das Attribut **start** zu.

Wiederholen Sie das Hinzufügen für das Feld **end**. Verlassen Sie den Dialog mit **OK** und beobachten Sie, wie das Anfangs- und das Enddatum des Vorgangs als Vorgangsbeschriftung erscheinen.



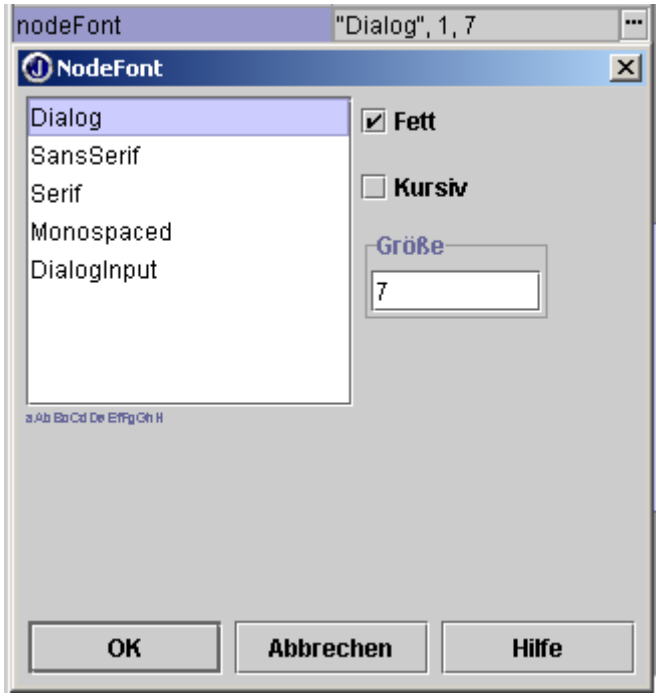
Sie haben mit den beiden eingegebenen Bezeichnungen die Attribute der Knoten-Entität spezifiziert, die die Anfangs- und Endtermine des Vorgangs enthalten. Aus diesem Grund werden diese Termine zur Laufzeit als Knotenbeschriftungen dargestellt.

Falls Sie - z.B. als Ihren ersten Versuch - die in einem der vorangegangenen Kapitel abgebildeten Knoten-Designs nachvollziehen möchten, richten Sie bitte mit Hinzufügen mindestens 10 Beschriftungsplätze ein (ohne diesen notwendigerweise Attribute aus einer Entität zuzuweisen). Nur dann werden in den Knotendesigns die Beschriftungsplatzhalter auch angezeigt.

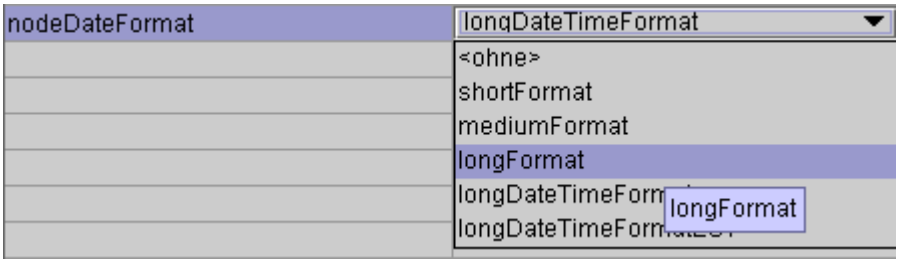


Die Position des Beschriftungsplatzes in der Liste entspricht dem Beschriftungsindex. Der oberste Platz entspricht dem Beschriftungsindex 1, der zweite dem Beschriftungsindex 2 usw. Sie können dies verifizieren, indem Sie einen Beschriftungsplatz markieren und seine Position mit den Tasten „Nach oben“ oder „Nach unten“ verändern. Beobachten Sie dabei die entsprechenden Veränderungen der Beschriftungen an den Knoten.

Wenn Sie Knotenbeschriftungen zuweisen, können Sie die Schrifteigenschaften und ein Format für Termine eingeben. Mit der Eigenschaft **nodeFont** können Sie Schriftart, Schriftschnitt und die Größe verändern. Klicken Sie bitte in das Feld rechts neben der Eigenschaft und anschließend auf die erscheinende “…” Taste, um den Dialog **NodeFont**, aufzurufen, den Sie bereits von der Tabelle und der Zeitskala kennen:

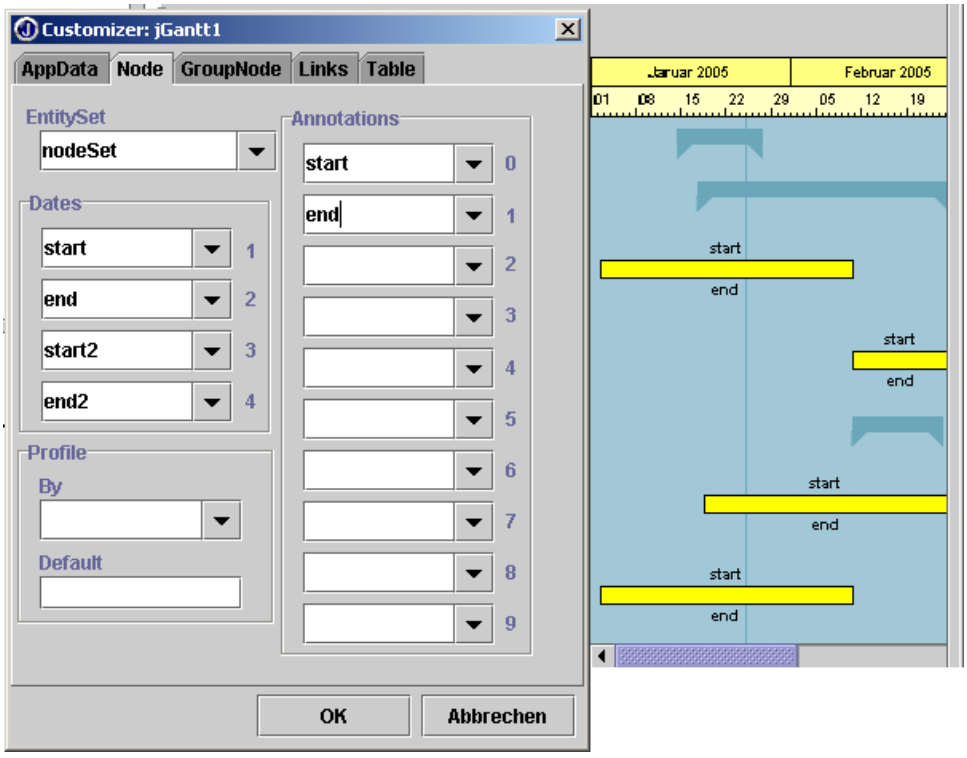


In diesem Dialog können Sie die genannten Einstellungen vornehmen. Um ein Datumsformat für Knotenbeschriftungen festzulegen, klicken Sie bitte in das Feld rechts der Eigenschaft NodeDateFormat und wählen Sie ein Datumsformat aus:



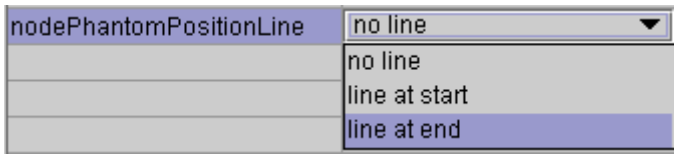
Die angebotenen Formate entstammen der Klasse **java.text.DateFormat**.

Sie können die Zuweisung von Knotenbeschriftungen auch im Customizer vornehmen. Klicken Sie die Kartei **Node** an, und wählen Sie das Entitäten-Set aus, für das die Knotenbeschriftung gesetzt werden soll, z.B. **nodeSet**. Im untenstehenden Beispiel wurden die Attribute **start** und **end** ausgewählt, um in den Positionen 0 und 1 des gewählten Knotenformats als Beschriftung angezeigt zu werden. Bei der Verwendung des Customizers haben Sie den Vorteil, dass die Position der Annotation direkt im Gantt-Graphen zur Design-Zeit angezeigt wird:

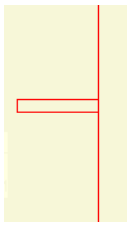


3.3.8 Die Positionierungslinie am Knotenphantom

Wenn der Benutzer einen Knoten verschiebt, werden seine Bewegungen von einem Knotenphantom verdeutlicht. Damit der Benutzer die Positionierung genau vornehmen und die Position des zu platzierenden Knotens gegen die Position vorhandener Knoten abgleichen kann, können Sie dem Phantom eine senkrechte Positionierungslinie an seinem Anfang oder Ende zuweisen:

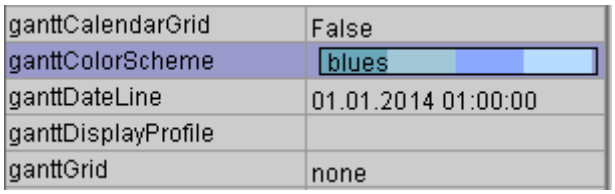


Wenn Sie zur Laufzeit einen Knoten mit der oben abgebildeten Einstellung verschieben, erscheint das Phantombild mit der Positionierungslinie am Ende:



3.4 Veränderungen am Gantt-Graphen

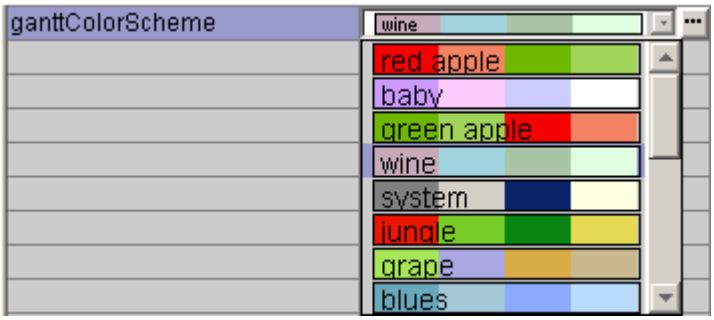
Die Eigenschaften, die sich auf den gesamten Gantt-Graphen auswirken, beginnen mit dem Namensbestandteil **gantt**.



3.4.1 Die Farbgebung des Gantt-Graphen

Auch dem Gantt-Graphen können Sie, wie anderen Elementen des Gantt-Diagramms, ein Farbschema zuweisen. Sie haben dies bereits beim Knoten

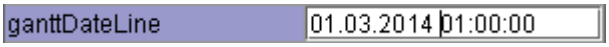
und bei der Zeitskala ausprobiert und sollten sich auch beim Gantt-Graphen ansehen, wie die vier Farben verwendet werden.



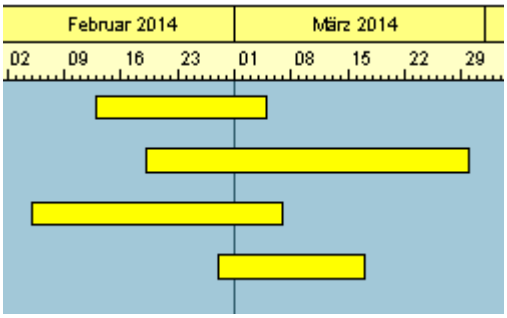
Die **shadedColor** wird für die Einfärbung des Hintergrundes im Gantt-Graphen verwendet, die **mainColor** färbt die Terminlinie (s. nächstes Kapitel) ein, während die **alternateColor** und die **shadedAlternateColor** bei der Darstellung eines Kalendergitters (ganttCalendarGrid) zur Verwendung kommen. Die lineColor färbt die Linien des Liniengitters ein (ganttGrid). Falls Sie mit den genannten Objekten zum jetzigen Zeitpunkt noch nicht vertraut sind, gehen Sie zunächst darüber hinweg. Sie dienen der Darstellung von zeitlichen Mustern im Gantt-Graphen und werden Ihnen in den weiteren Kapiteln noch vorgestellt.

3.4.2 Die Terminlinie

Gitter und Profile werden Sie in einem späteren Kapitel kennen lernen, aber eine Terminlinie können Sie jetzt schon setzen. Sie dient dazu, z.B. einen bestimmten Termin, z.B. einen Stichtag optisch zu markieren. Geben Sie dazu bitte ein Datum in das Feld **ganttDateLine** ein:



Achten Sie bitte darauf, dass das eingegebene Datum zwischen Anfangs- und Enddatum Ihrer Zeitskala liegt, sonst wird die Stichtagslinie nicht sichtbar:



3.5 Veränderungen an der Tabelle

Um Veränderungen an der Tabelle vorzunehmen, scrollen Sie zu den Eigenschaften, die mit dem Namensbestandteil **table** beginnen. Mit den vorhandenen Eigenschaften können Sie sie nach Ihren Wünschen gestalten:

tableColorScheme	system
tableColumns	
tableColumnTitleFont	
tableColumnTitlesSource	entity attribute display names
tableColumnWidths	15.0<NL>25.0<NL>18.0<NL>
tableDateFormat	
tableEditable	False
tableFont	"Arial", 0, 8
tableGroupFont	
tableHierarchyColumn	2
tableRowTitleFont	

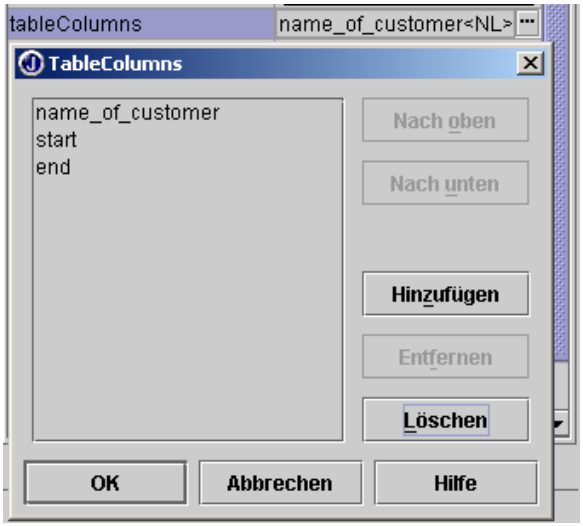
3.5.1 Die Festlegung der Tabellenspalten

Die Größe der Tabelle wird durch ihre Anzahl an Zeilen und Spalten bestimmt. Während die Anzahl der Zeilen durch die Anzahl der existierenden Knoten vorgegeben ist, können Sie die Anzahl der Spalten über eine Eigenschaft festlegen.

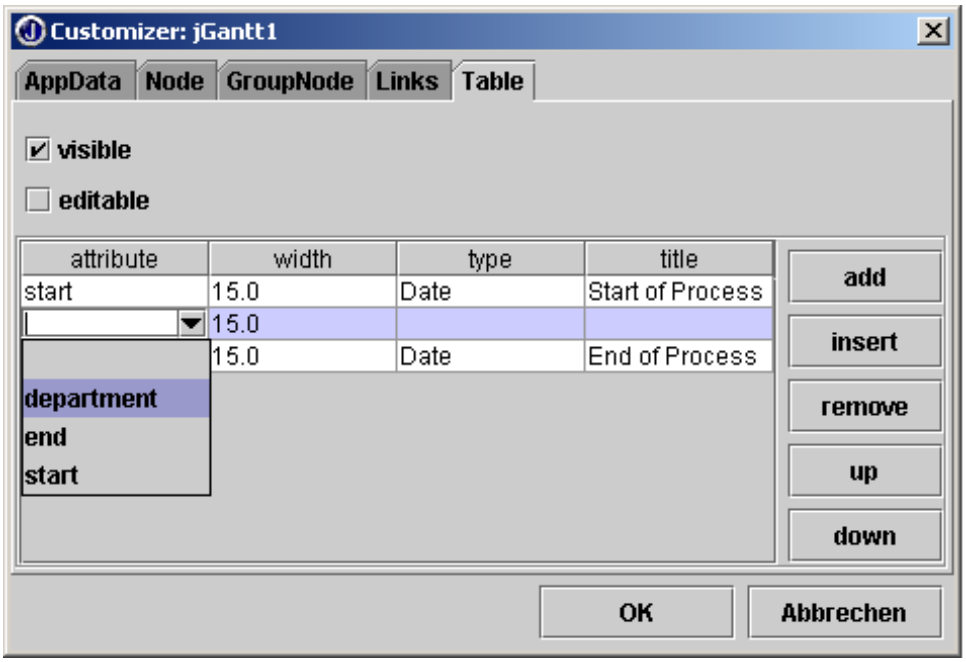
Klicken Sie dazu in das Feld rechts neben der Eigenschaft **tableColumns** und anschließend in das Feld mit den drei Punkten. Sie gelangen zu dem entsprechenden Dialog, in dem Sie durch Hinzufügen eine Spalte hinzufügen. Anschließend weisen Sie der angelegten Spalte den Namen eines Entitätsattributes zu, indem Sie den Namen durch Klicken editieren und durch den Namen eines existierenden Attributs ersetzen.

In unserem Beispiel unten haben wir drei Spalten angelegt und mit den existierenden Attributen **start**, **end** und **name_of_customer** belegt. Auf diese Weise können Sie einen beliebigen Teil der Knotendaten in der Tabelle wiedergeben.

Sie können die Reihenfolge der Spalten in der Tabelle über die Position der Namen im Dialog unten steuern. Der oberste Name im Dialog entspricht der äußersten linken Spalte des Tabellenrumpfes. Mit den Schaltflächen **Nach oben** und **Nach unten** lassen sich die Positionen verändern, sobald Sie einen Namen markiert haben.



Sie können die Tabelle auch mit dem Customizer bearbeiten. Nach dem Aufruf aktivieren Sie die Karteikarte **Table**. Hier können Sie einstellen, ob die Tabelle sichtbar (visible) und editierbar (editable) sein soll. Weiterhin können Sie hier über die Schaltflächen insert und **add** weitere Spalten oberhalb des Cursors bzw. am Ende der Tabelle einrichten. Anschließend wählen Sie die Attribute aus, deren Daten in der Spalte angezeigt werden sollen:

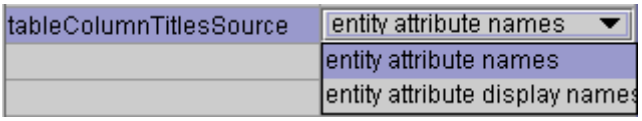


Während Sie die Felder **attribute** und **width** (Spaltenbreite) editieren können, dienen die restlichen Felder nur der Anzeige und sind auf dieser Karteikarte unveränderbar.

Mit der Schaltfläche **remove** können Sie markierte Spalten löschen. Mit den Tasten up und down können Sie die Reihenfolge der Attribute und damit der Spalten verändern. Dabei entspricht die oberste in diesem Dialog angezeigte Zeile der linken Spalte.

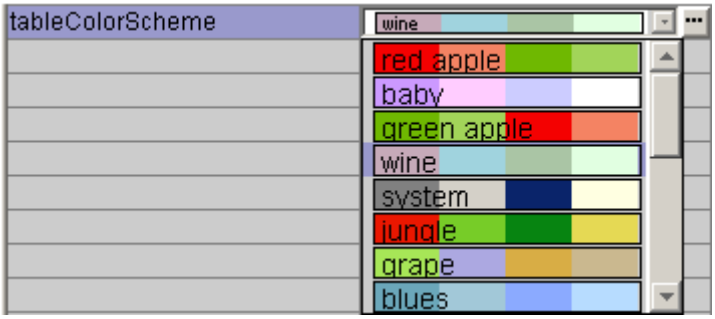
3.5.2 Die Festlegung der Quelle für Spaltenüberschriften

Sie können über die Eigenschaft **tableColumnTitlesSource** die Quelle bestimmen, aus der die Spaltenüberschriften eingelesen werden. Sie haben die Wahl zwischen den tatsächlichen Attributnamen der Knoten-Entität (entity attribute names) oder zwischen in eigenen Attributen angelegten Namen (display names):



3.5.3 Die Farbgebung der Tabelle

Auch der Tabelle können Sie, wie anderen Elementen des Gantt-Diagramms, ein Farbschema zuweisen. Sie haben dies bereits in den vorausgehenden Kapiteln ausprobiert und sollten sich auch bei der Tabelle ansehen, wie die vier Farben verwendet werden.

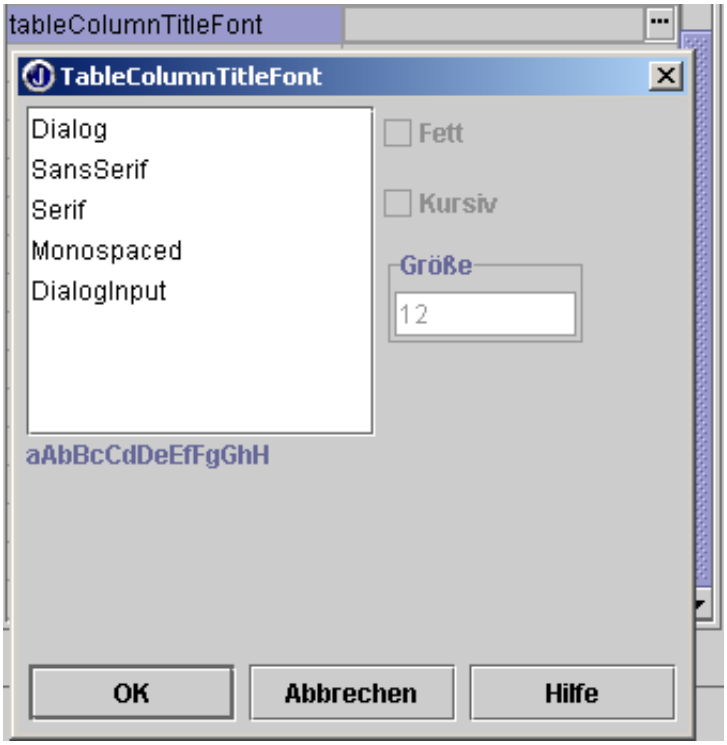


Die mainColor wird für die Einfärbung der äußersten linken Spalte verwendet. Die shadedColor gibt dem Tabellenrumpf Farbe, die **alternateColor** färbt die oberste Zeile ein, während schließlich die **shadedAlternateColor** für die Farbe der Gruppenzeilen verantwortlich ist.

3.5.4 Festlegung von Schriftarten in der Tabelle

In der Tabelle können Sie für vier verschiedene Bereiche die Schriftarten eigens festlegen:

- Für den Tabellenrumpf, dafür wählen Sie die Eigenschaft **tableFont**.
- Für die Spaltenüberschrift (oberste waagerechte Zeile), dafür wählen Sie **tableColumnTitleFont**.
- Für Gruppenüberschriften (Zeilen, in denen Gruppendaten zusammengefasst sind), dafür wählen Sie **tableGroupFont**.
- Für die Zeilenüberschrift (äußerste linke Spalte, ohne oberstes Feld), dafür wählen Sie **tableRowTitleFont**:
- Klicken Sie auf das Feld rechts neben der jeweiligen Eigenschaft und dann auf das erscheinende Feld mit den drei Punkten. Es erscheint dieser oder ein ähnlicher Dialog.



Stellen Sie verschiedene Schriftarten ein und beobachten Sie die Veränderungen in der Tabelle.

Neben der Einstellung über die Eigenschaften bietet VARCHART JGantt auch die völlig freie Programmierung der Tabelle, über die Sie mehr in den folgenden Kapiteln erfahren.

4 Gantt-Anwendung neu entwickeln

Sie werden in diesem Kapitel Ihre Kenntnisse über die grundlegenden Aspekte zum Programmieren Ihres Gantt-Diagramms vertiefen, wobei Sie Source-Code überwiegend manuell erstellen. Zunächst werden wir eine neue Applikation anlegen und Schritt für Schritt die Programmierung des Gantt-Diagramms aufzeigen.

Bei den folgenden Beispielen sind einige allgemeine Gegebenheiten spezifisch für die verwendete Programmierumgebung. Wenn Sie mit Ihrer JAVA Programmierumgebung vertraut sind, sollte es unproblematisch sein, diese Gegebenheiten dorthin zu übertragen. Der spezielle VARCHART JGantt-Teil ist unabhängig von der Programmierumgebung.

4.1 Automatisch erzeugter Quellcode

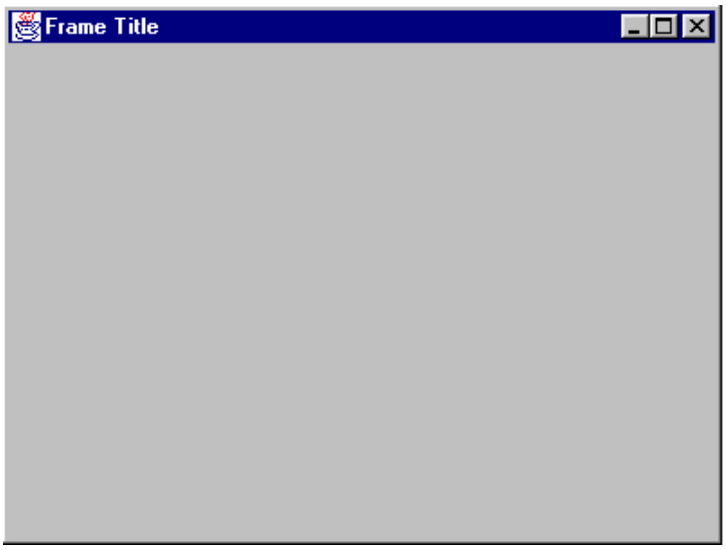
Wenn Sie eine neue Applikation mit einem Fenster anlegen, generieren die Umgebungen meist automatisch zwei Klassen. Die erste heißt **Frame1** und richtet ein Fenster ein, die zweite heißt **Application1** und enthält die Main-Methode, die das Fenster positioniert und auf den Bildschirm bringt. Das Package, dem die Klassen angehören, heißt - sofern Sie keinen anderen Namen vergeben -automatisch **unbenannt1**. Hier der Source-Code der beiden Programme:

```
package unbenannt1;import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Frame1 extends JFrame
{
JPanel contentPane;
BorderLayout borderLayout1 = new BorderLayout();
   /**Construct the frame*/
   public Frame1()
      {
      enableEvents(AWTEvent.WINDOW_EVENT_MASK);
      try
         {
         jbInit();
         }
      catch(Exception e)
         {
         e.printStackTrace();
         }
      }
```

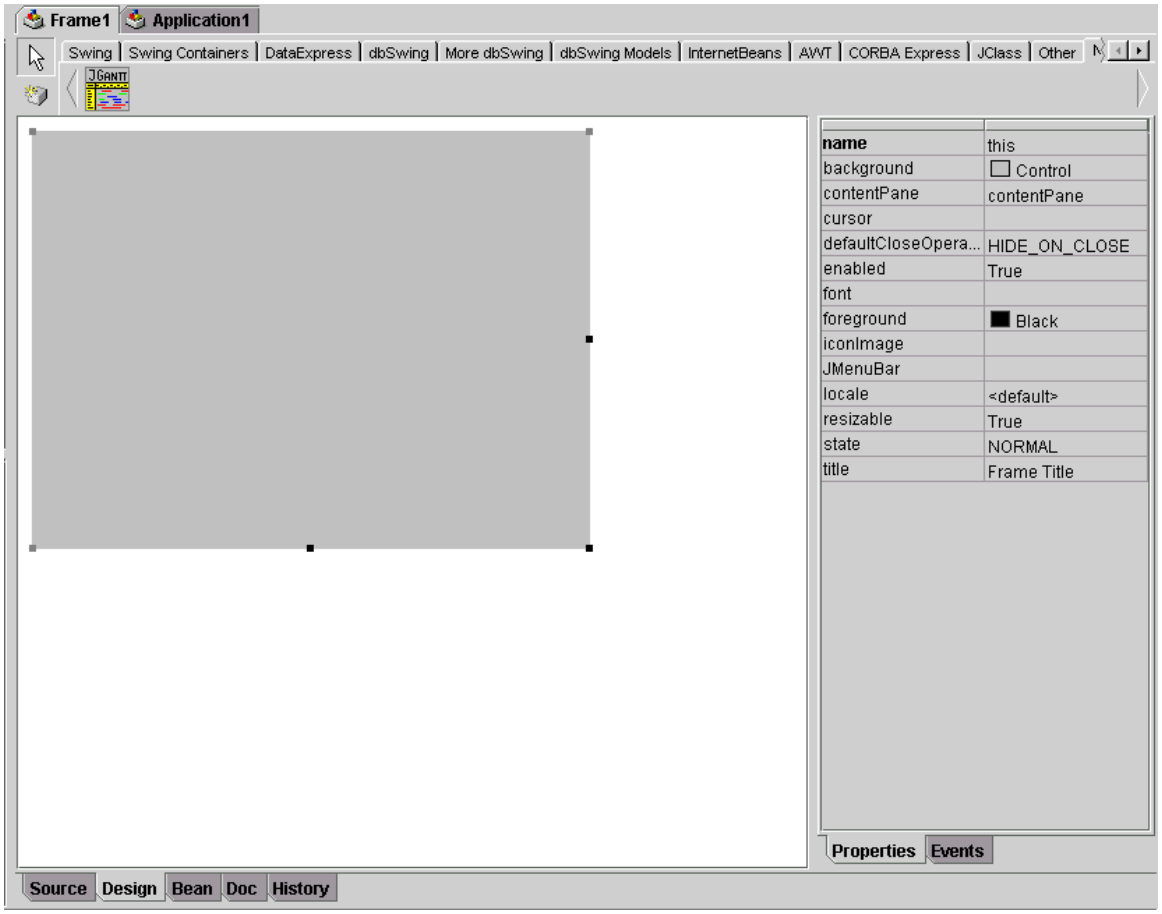
```
 /**Component initialization*/
 private void jbInit() throws Exception
    {
    contentPane = (JPanel) this.getContentPane();
    contentPane.setLayout(borderLayout1);
    this.setSize(new Dimension(400, 300));
    this.setTitle("Frame Title");
    }

/**Overridden so we can exit when window is closed*/
protected void processWindowEvent(WindowEvent e)
   {
   super.processWindowEvent(e);
   if (e.getID() == WindowEvent.WINDOW_CLOSING)
      {
      System.exit(0);
      }
   }
}

package unbenannt1;
import javax.swing.UIManager;
import java.awt.*;

public class Application1
{
boolean packFrame = false;

   /**Constructor*/
   public Application1()
   {
   Frame1 frame = new Frame1();

    //Validate frames that have preset sizes
    //Pack frames that have useful preferred size info, e.g. from
    //their layout
    if (packFrame)
       {
       frame.pack();
       }
    Else
       {
       frame.validate();
       }

    //Center the window
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = frame.getSize();
    if (frameSize.height > screenSize.height)
        frameSize.height = screenSize.height;

    if (frameSize.width > screenSize.width)
        frameSize.width = screenSize.width;

    frame.setLocation((screenSize.width - frameSize.width) / 2,
                      (screenSize.height - frameSize.height) / 2);
    frame.setVisible(true);
    try
       {
       jbInit();
```

```
        }
     catch(Exception e)
        {
        e.printStackTrace();
        }
      }

     /**Main method*/
     public static void main(String[] args) {
     try
        {
        UIManager.setLookAndFeel(
                UIManager.getSystemLookAndFeelClassName());
        }
     catch(Exception e)
        {
        e.printStackTrace();
        }
      new Application1();
    }
    private void jbInit() throws Exception
    {
    }
}
```

Das Ergebnis ist ein leeres Fenster:



Wir setzen voraus, dass Sie mit dem Programmieren in Java vertraut sind und gehen nicht weiter auf den obenstehenden Code ein. Wenn Sie in Ihrem grafischen Editor (Design-Mode) die Klasse **Frame1** öffnen und das **Netronic**-Tab aktivieren, sehen Sie zunächst ihre grau unterlegte Darstellung des leeren Fensters:



Sobald Sie nun die JGantt-Ikone in das Fenster hineinziehen, erhalten Sie folgende Änderungen im Source-Code:

```
import de.netronic.jgantt.*;
```

Zu den Import-Statements wurde ein weiteres addiert: die Klasse **de.netronic.jgantt** wird zusätzlich importiert. Weiterhin wird ein neues jGantt-Objekt mit dem Namen **jGantt1** angelegt.

```
public class Frame1 {
    JGantt jGantt1 = new JGantt();
...
```

Bevor wir das Programm weiterentwickeln, vorab einige Grundsätzlichkeiten zunächst zur Lizenzierung, die über das JGantt-Objekt erfolgt, und anschließend zu Daten in VARCHART JGantt.

4.2 Die Lizenzierung des VARCHART JGantt JavaBean-Objektes

Die Lizenzierung von VARCHART JGantt JavaBean erfolgt über einen Lizenz-Code, den Sie bei NETRONIC Software käuflich erwerben können. Mit ihm heben Sie den Zeitschutz auf, der in der Software implementiert ist und der ihre Funktion nach spätestens 3 Monaten unentgeltlicher Testzeit sperrt.

4.2.1 Arten der Lizenzierung

Je nach Art der Verwendung von VARCHART JGantt können Sie verschiedene Lizenzen erwerben, für die NETRONIC jeweils entsprechende Informationen von Ihnen benötigt:

a) Sie können eine Lizenz für einen bestimmten Rechner erwerben, dazu benötigt NETRONIC zu dessen Identifizierung die MAC-Adresse.

b) Sie können eine Lizenz für einen bestimmten Benutzer erwerben, dazu benötigt NETRONIC den Namen dieses Benutzers im Netzwerk.

c) Sie erwerben eine Firmenlizenz, dazu benötigt NETRONIC Ihren Firmennamen.

d) NETRONIC benötigt außerdem die Versionsnummer der JGantt JavaBean Edition (normalerweise die aktuelle Version), die Sie einsetzen wollen.

e) Zudem sollten Sie wissen, ob Sie Daten nur anzeigen oder auch editieren wollen

f) .... und ob Sie die Histogramm-Funktionalität benötigen.

Die Daten zu a) und b) können Sie mit Hilfe des mitgelieferten Tools **ComputerIdentification.jar** im Unterverzeichnis „Tools“ bestimmen.

Für Testzwecke können Sie auch eine zeitlich begrenzte Lizenz erwerben.

4.2.2 Die Verwendung einer Lizenz

Sie erhalten von NETRONIC einen Lizenz-String und eine Lizenzdatei.

Den Lizenz-Code können Sie als Parameter bei der Generierung des JGantt-Objektes übergeben:

```
JGantt jGantt1 = new JGantt("220451,797420,923401;JGantt 2.3;Firma Hans
Mustermensch;090331;MACADDR=00-11-D8-98-90-C4");
```

In dem obigen Beispiel heben Sie den Zeitschutz auf und Ihre VARCHART-JGantt- Applikation wird auf einem Rechner mit der angegebenen MAC-Adresse zeitlich unbegrenzt lauffähig. Ist der Lizenz-Code ungültig, können Sie VARCHART JGantt JavaBean trotz evtl. noch gültiger Demo-Lizenz nicht benutzen.

Alternativ geben Sie den Lizenz-String nicht im Konstruktor an, dann (und nur dann) sucht JGantt automatisch nach der Datei **JGantt.lic**, die den Lizenzierungs-Code enthält. Diese wird im aktuellen Verzeichnis gesucht: Das aktuelle Verzeichnis ist dasjenige, in welchem die jar-Datei Ihrer Applikation liegt. Wenn Sie Ihre Applikation von einer Entwicklungsoberfläche aus starten, legt diese das aktuelle Verzeichnis fest.

Wird eine lesbare Datei mit dem Namen **JGantt.lic** nicht im aktuellen Verzeichnis gefunden, wird im Home-Verzeichnis des Benutzers weitergesucht. Sobald eine lesbare Datei **JGantt.lic** gefunden wurde, wird diese verwendet, selbst wenn der Lizenz-Code darin ungültig ist. In einem solchen Fall schlägt die Lizenzierung fehl, selbst wenn die Demo-Lizenz noch gültig ist.

Hinweise:

Die Lizenzierung über eine MAC- Adresse ist derzeit nur für Windows, Linux und MacOS möglich; sollten Sie darüber hinaus Bedarf haben, kontaktieren Sie bitte den Support unter

netronic.de > Produkte für Entwickler > VARCHART JGantt > Support

oder:

VARCHART-JGantt-Support

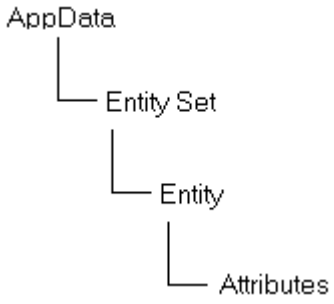
4.3 Datenstrukturen

Die Vereinbarung der benötigten Datenstrukturen ist bei der Programmierung eines VARCHART JGantt-Anwendung von grundlegender Bedeutung. Die Datendefinition nehmen Sie jetzt, ebenso wie das Füllen der Datenstrukturen mit Inhalt, über die Programmierschnittstelle im Quellcode vor.

Grundsätzlich können Sie bei der Anlage von Daten wählen, ob Sie die Daten zur Laufzeit über die API generieren, oder ob Sie sie über eine XML-Datei einlesen bzw. speichern.

In unserem Beispiel werden die Definition der Datenstruktur sowie ihr Inhalt in einer Datei im XML-Format auf einem Speichermedium abgelegt, um ggf. zu einem späteren Zeitpunkt wieder eingelesen zu werden.

Für die Datendefinition verwenden Sie die Komponente **AppData**. Sie stellt die Daten für Ihre Applikation bereit. Die Komponente enthält eine Hierarchie von Objekten, mit deren Hilfe Sie die für Ihre Applikation notwendigen Daten strukturieren können:



Das zentrale Element in der Objekthierarchie ist die Entität. Sie entspricht dem Datensatz, in dem Sie alle Daten eines Vorgangs ablegen würden. Aus unserem Beispiel wären dies die Daten eines Zimmerbelegungsvorgangs. Eine Entität besteht aus Attributen, vergleichbar den Datensatzfeldern, z.B. Zimmernummer, der An- oder Abreisetermin eines Gastes u.a. Entitäten sind zu einem Entity Set zusammengefasst, ähnlich einer Tabelle, und von allen Entitäten-Sets gibt es eine Liste oder Sammlung aller Applikationsdaten (application data, abgekürzt **AppData**, was einer Datenbank entspricht. Der Unterschied zwischen den beiden analogen Strukturelementen ist der, dass in den Entitäten-Objekte gespeichert werden, während in einem Datensatz einfache Daten abgelegt werden.

4.4 Das Anlegen von Datenstrukturen

Da VARCHART JGantt für die Darstellung des Diagramms im Design-Mode eine automatische, minimale 0Initialisierung vornimmt, werden auch bereits einige Defaultwerte aus der **appData**-Komponente gesetzt, z.B. die Vorgangstermine. Standardmäßig sind vier Termine pro Vorgang vorgesehen. Diese Vorgangstermine können Sie im Designmode im Pane beim Menüpunkt **NodeDates** verändern:

Setzen Sie die ersten beiden Defaultwerte einmal auf die im Bild gezeigten Werte. Durch Doppelklick auf den Namen können Sie diesen editieren. Wenn Sie nun in den Quellcode-Modus wechseln, finden Sie in der Initialisierung folgende Befehlszeile:

```
jGantt1.setNodeDates(new String[] {"start", "end", "start2", "end2"});
```

Mit diesem Aufruf benennen Sie die vier vorhandenen Datenattribute des Vorgangs. Sie können die Anzahl dieser Attribute derzeitig nicht ändern. Für zukünftige Versionen ist dies allerdings vorgesehen. Die Methode wird von einem VARCHART JGantt Package zur Verfügung gestellt, das Sie am Programmanfang entsprechend importieren:

Als nächstes benennen Sie das Entitäten-Set, das die Daten über die Vorgänge nthält. Nennen Sie es beispielsweise **Rooms**:

```
jGantt1.setNodeSetName("Rooms");
```

Um die Tabelle mit sinnvollen Werten zu füllen, geben Sie jetzt die Entitäten-Attribute des Vorgangs an, die in der Tabelle erscheinen sollen:

```
jGantt1.setTableColumns(new String[]{"name_of_customer","start","end"});
```

Hier nun ein Anweisungsblock im Überblick, der im Folgenden erläutert wird:

```
// Datendefinition
NeIAppData appData = jGantt1.getAppData();
NeIEntitySet entitySet = appData.createEntitySet("Rooms");
entitySet.addEntityAttribute("no_of_room", String.class, "room
number");
entitySet.addEntityAttribute("start", Date.class, "start of stay");
entitySet.addEntityAttribute("end", Date.class, "end of stay");
```

```
entitySet.addEntityAttribute("amount", Double.class, "amount to pay");
entitySet.addEntityAttribute("name_of_customer" String.class,
                             "customer name");
appData.getXMLInterface().saveXML("c:\\RoomData.xml");
```

Erläuterungen:
Die NETRONIC Interfaces (NeI) gehören dem Package **de.netronic.-common.intface** an, das Sie am Programmanfang entsprechend importieren:

```
import de.netronic.common.intface.*;
```

Für die Definition Ihrer Applikationsdaten holen Sie zunächst eine Instanz der JGantt-AppData:

```
NeIAppData appData = jGantt1.getAppData();
```

Als Nächstes legen Sie ein Entitäten-Set mit dem Namen „Rooms“ an:

```
NeIEntitySet entitySet = appData.createEntitySet("Rooms");
```

Anschließend füllen Sie diese Liste mit Attributen, die Ihre Vorgänge beschreiben. Diese sollen beispielhaft zunächst die Zimmernummer (no_of_room), das Anfangsdatum der Zimmerbelegung (start), das Ende der Zimmerbelegung (end), der Preis (amount) und der Name des Kunden (name_of_customer) sein.

Für die Deklaration des Datums benötigen Sie vorab die Klasse java.util.Date, die Sie am Programmanfang importieren:

```
import java.util.Date;
```

Die Namen der Attribute werden in der Methode addEntityAttribute als erster Parameter übergeben. Im zweiten Parameter weisen Sie den Attributen ein Datentyp-Objekt zu. VARCHART-JGantt bietet vier verschiedene Datentyp-Objekte: das Integer-Objekt, das Datums-Objekt, das Double-Objekt und das String-Objekt. Sie werden mit der Erweiterung ".class" deklariert. Im dritten Parameter wird eine sinnvolle Beschreibung übergeben.

```
entitySet.addEntityAttribute("no_of_room", String.class, "room number");
entitySet.addEntityAttribute("start", Date.class, "start of stay");
entitySet.addEntityAttribute("end", Date.class, "end of stay");
entitySet.addEntityAttribute("amount", Double.class, "amount to pay");
entitySet.addEntityAttribute("name_of_customer", String.class,
                             "customer name");
```

Zu Übungszwecken legen Sie das Ergebnis zunächst in einer Datei mit dem Namen "RoomData" ab:

```
appData.getXMLInterface().saveXML("c:\\RoomData.xml");
```

Sie haben nun neue Attribute der Liste der Attribute hinzugefügt. Bevor wir nun das Ergebnis in der XML-Datei anschauen, hier noch ein Hinweis zu den

Datentypen: Alle Datentypen werden intern als Datentyp-Objekt behandelt und als solches weitergegeben. Wollen Sie bei einem **getValue**-Aufruf einen reinen Datentyp erhalten, verwenden Sie die Aufrufe:

```
getValueAsInteger
getValueAsString
getValueAsDouble
getValueAsDate
```

Damit erzwingen Sie die Umwandlung von einem Datentyp-Objekt in einen Datentypen.

Zurück zu unserem Beispiel. Nach dem Schreiben der Datenstruktur in die XML-Datei erhalten Sie folgendes Ergebnis:

```
<?xml version="1.0" encoding="UTF-8"?>
<AppData>
  <schema>
    <complexType name="Data">
      <complexType name="Rooms">
        <attribute name="systemID" type="long"/>
        <attribute name="userID" type="string"/>
        <attribute name="op" type="string"/>
        <element name="no_of_room" type="" displayName="room number"/>
        <element name="start" type="" displayName="start of stay"/>
        <element name="end" type="" displayName="end of stay"/>
        <element name="amount" type="" displayName=amount to pay""/>
        <element name="name_of_customer" type="" displayName= "customer
name"/>
    </complexType>
   </complexType>
  </schema>
  <Data>
  </Data>
</AppData>
```

Erläuterung:
Die XML-Datei ist durch sogenannte **Tags** gekennzeichnet, die die abgespeicherten Daten charakterisieren. Tags bestehen aus einem Schlüsselwort und Parametern, die in eine Anfang- und Ende-Markierung gefasst sind. Sie bestehen aus einer spitzen Klammer „ < “ bzw. einem Schrägstrich kombiniert mit einer spitzen Klammer „ /> “. Tags können geschachtelt werden, in solchen Fällen gibt es ein Anfangs- und ein End-Tag. Das Anfangstag besteht aus einem Schlüsselwort und zwei spitzen Klammern <Schlüsselwort>, das Endtag aus zwei spitzen Klammern mit einen Slash vor dem Schlüsselwort </Schlüsselwort>. Innerhalb der spitzen Klammern können außer dem Tag-Namen auch Parameter stehen. Das oberste XML-Tag enthält eine Angabe zum verwendeten Zeichensatz:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Das Tag <AppData> kennzeichnet eine zu VARCHART JGantt gehörige Datei, die die Applikationsdaten enthält. Das Tag <schema> ist ein name space, der für die eindeutige Definition der gelieferten Tags benutzt wird. Das folgende Tag (zusammen mit seinem schließenden Tag) deklariert, dass die Daten in einem Abschnitt liegen sollen, der von dem Tag <Data> </Data> eingeschlossen ist:

```
<complexType name="Data">
```

Das folgende Tag/End-Tag beschreibt den Namen einer Untereinheit im Abschnitt "Data" von "Rooms":

```
<complexType name="Rooms">
```

Das Tag "attribute" weist der Untereinheit eine Systemnummer, einen Benutzernamen und eine Operation mit den entsprechenden Datentypen zu:

```
<attribute name="systemID" type="long"/>
<attribute name="userID" type="string"/>
<attribute name="op" type="string"/>
```

Mit dem Tag „element“ werden die eigentlichen Attributdefinitionen vorgenommen. Dabei werden der Name, der Datentyp (der im Falle von Datenobjekten leer bleibt) und der Kommentar abgelegt:

```
<element name="no_of_room" type="" displayName="room number"/>
<element name="start" type="" displayName="start of stay"/>
<element name="end" type="" displayName="end of stay"/>
<element name="amount" type="" displayName=amount to pay""/>
<element name="name_of_customer" type="" displayName="customer name"/>
```

Sie haben nun die XML-Datendefinition für **Rooms** erstellt. Knoten im Gantt-Diagramm werden aber erst sichtbar, wenn Daten existieren. Daher besteht der nächste Schritt darin, Daten für die erstellten Strukturen anzulegen.

4.5 Das Anlegen von Daten

Nach der Strukturierung der Daten legen Sie nun die Daten in den Strukturen an. Hier ein Anweisungsblock, der im Folgenden erläutert wird:

```
// Anlegen von Daten
NeIEntity entity = entitySet.createEntity("");
entity.setValue("no_of_room", 3);
Calendar cal = new GregorianCalendar();
cal.set(2005, 4, 2, 18, 0, 0);
entity.setValue("start", cal.getTime());
cal.set(2005, 4, 8, 12, 30, 0);
entity.setValue("end", cal.getTime());
```

```
entity.setValue("amount", 65.50);
appData.getXMLInterface().saveXML("c:\\RoomData.xml");
```

Erläuterung:

Zunächst legen Sie eine Entität an und initialisieren diese mit Leerzeichen:

```
NeIEntity entity = entitySet.createEntity("");
```

Anschließend weisen Sie dem Attribut **no_of_room** eine Zimmernummer zu:

```
entity.setValue("no_of_room", 3);
```

Nun legen Sie ein Datum an, das Sie dem Attribut **start** zuweisen. Dafür importieren Sie zunächst am Programmanfang folgendes Package:

```
import java.util.GregorianCalendar;
```

Dann legen Sie eine Instanz des Gregorianischen Kalenders in die Variable **cal** an:

```
Calendar cal = new GregorianCalendar();
```

Diese stellt Ihnen eine Set-Methode zur Verfügung, mit der Sie das Datum (2. April 2012 18:00:00 Uhr) anlegen sowie eine **GetTime**-Methode, mit der Sie es irgendwo eintragen können. Mit der Methode **setValue** der Entität weisen Sie das Datum in der Form eines Objektes dem Attribut **start** zu:

```
cal.set(2012, 4, 2, 18, 0, 0);
entity.setValue("start", cal.getTime());
```

Das Anlegen und Zuweisen eines Datums wiederholen Sie für das Enddatum des Belegungsvorgangs. Auf die gleiche Weise wie die Zimmernummer füllen Sie nun noch den Preis. Anschließend legen Sie die Daten in der XML-Datei ab:

```
appData.getXMLInterface().saveXML("c:\\RoomData.xml");
```

In der XML-Datei wurde nun neben der Datendefinition auch der Datenbereich gefüllt:

```
…
<Data>
   <Rooms systemID="1" userID="" op="insert">
      <no_of_room>3</no_of_room>
      <start>2012-05-02T18:00:00.926</start>
      <end>2012-05-08T12:30:00.926</end>
      <amount>65.5</amount>
   </Rooms>
</Data>
```

Wenn Sie das erste Entitäten-Set füllen, dient dieses als Template für alle weiteren.

In diesem Kapitel haben Sie Datenstrukturen vereinbart und Daten angelegt. Im nächsten Kapitel lernen Sie, wie Sie das Einfügen von Daten feststellen und darauf reagieren können.

4.6 Ereignisse der AppData-Komponente

Wenn der Benutzer im Gantt-Diagramm einen Knoten aufzieht, d.h. einen Vorgang generiert, werden automatisch nur die Daten in die Entität eingetragen, die seinen Anfang und sein Ende begrenzen. Eine Entität umfasst aber in der Regel mehr Attribute als diese beiden und dem Benutzer sollte daher die Möglichkeit gegeben werden, weitere Daten zu ergänzen, z.B. über eine Dialogbox den Namen des Kunden manuell eingeben zu können. Aus solchen Gründen müssen Sie in der Lage sein, auf Änderungen am Entitäten-Set zu reagieren.
Wir werden in diesem Kapitel den Namen des Kunden interaktiv erfragen, um ihn der Entität hinzuzufügen und in der XML-Datei abzulegen.

Um Änderungen an einem Entitäten-Set festzustellen, können Sie eine Klasse bei der Komponente **AppData** als EventListener registrieren. Sie erhalten dann Meldungen darüber, ob eine Entität hinzugefügt, verändert oder gelöscht wurde. Wenn Sie diese Änderung feststellen, können Sie bei einer Löschung z.B. mit einer Sicherheitsabfrage oder bei einer Neuanlage mit einer Dialogbox reagieren, die dem Benutzer anbietet, die fehlenden Daten einzugeben.

Zunächst importieren Sie am Programmanfang die Packages:

```
import de.netronic.jgantt.*;
import de.netronic.common.intface.*;
import de.netronic.common.event.*;
```

Das JGantt-Package macht den Namensraum des JGantt-Objektes bekannt; das intface-Package wird für die AppData benötigt und das event-Package stellt Ihnen den Namensraum für die Klasse NeAppDataAdapter zur Verfügung.

Sie verwenden nun die Klasse NeAppDataAdapter, um davon eine neue Klasse abzuleiten und mit Hilfe dieser die vorhandenen Methoden zu überschreiben. Insgesamt existieren dort sechs Methoden, von denen wir drei hier beispielhaft überschreiben, und zwar die Methoden zum Anlegen, Löschen und Ändern einer Entität:

```
public class AppDataEventListener extends NeAppDataAdapter
{
   public void entityCreated (NeAppDataEvent evt)
     {
        …
     }
   public void entityDeleted (NeAppDataEvent evt)
     {
        …
     }
   public void entityChanged (NeAppDataEvent evt)
     {
       …
     }
 }
```

In der Initialisierungsphase Ihres Programms teilen Sie dem EventListener mit, dass die neu geschriebene Klasse die Adresse ist, an die die Ereignisse geschickt werden (hier: AppDataEventListener) und von welchem EntitySet die Ereignisse gemeldet werden sollen (Rooms). Den dritten Bool'schen Parameter setzen Sie auf **true**, wenn für alle bereits vorhandenen Entitäten ein EntityCreated-Event geschickt werden soll, jedoch auf **false**, wenn er nur bei neuen geschickt werden soll:

```
appData.addAppDataEventListener (new AppDataEventListener(), "Rooms",
false);
```

Um für eine spätere Verarbeitung geänderte Daten vorzuhalten, schalten Sie nun das Logging ein. Es wird für das eigentliche Event-Handling nicht benötigt, sondern dient dazu, alle Änderungen in der **AppData**-Komponente im XML-Format zu protokollieren, um sie später erfragen zu können:

```
appData.getXMLInterface().setXMLLogging(NeIXML.NE_FULL_LOGGING);
```

Jetzt implementieren Sie eine Behandlung für die in der Klasse AppDataEventListener überschriebenen Methoden. In unserem Beispiel besteht die Behandlung darin, die in der Log-Datei festgehaltenen Änderungen der Entität auszudrucken sowie bei der Neuanlage eines Knotens in einem Standard-Dialog den Namen des Kunden zu erfragen. Hier der Anweisungsblock, der im folgenden erläutert wird:

```
public void entityCreated (NeAppDataEvent evt)
{
  //Namen erfragen
  String customerName = JOptionPane.showInputDialog("Please
                                              enter name of customer");
  evt.getEntity().setValue("name_of_customer", customerName);
  getXmlInfo();
}

public void entityDeleted (NeAppDataEvent evt)
```

```
{
  getXmlInfo();
}

public void entityChanged (NeAppDataEvent evt)
{
  getXmlInfo();
}

private void getXmlInfo (NeAppDataEvent evt)
{
   //Holen der AppData
   NeIAppData appData = evt.getEntitySet().getAppData();

   //Ausgabe des XML formatierten Log-Strings
   System.out.println(appData.getXMLInterface().getXMLLogString()+"\n");

   //Neuinitialisierung des Log-Strings
   appData.getXMLInterface().setXMLLogString("");

   //Speichern der AppData als XML Datei
   appData.getXMLInterface().saveXML("RoomData.xml");
}
```

Erläuterung:
Die entsprechenden Methoden des Event-Listeners werden aufgerufen, und zwar für das Hinzufügen einer Entität die Funktion **entityCreated**. Sie erhält einen Parameter **evt** vom Typ **NeAppDataEvent**:

```
private void entityCreated (NeAppDataEvent evt)
```

Zunächst wird eine Meldung ausgedruckt. Dabei werden die protokollierten Änderungen als XML-formatierte Zeichenkette ausgegeben:

```
NeIAppData appData = jGantt1.getAppData();
System.out.println(appData.getXMLInterface().getXMLLogString()+"\n");
```

Nachdem die Daten des Log-Files ausgewertet wurden, wird der Log-Buffer mit Leerzeichen neu initialisiert:

```
appData.getXMLInterface().setXMLLogString("");
```

Bei der Neuanlage einer Entität wird nun der Name des Kunden vom Benutzer mit Hilfe einer Standard-Dialogbox als zusätzliche Datenmenge in die Entität aufgenommen:

```
String customerName = JOptionPane.showInputDialog("Please
                                                   enter name of customer");
evt.getEntity().setValue("name_of_customer", customerName);
```

Mit dem folgenden Aufruf wird die gesamte appData in der XML-Datei abgespeichert:

```
appData.getXMLInterface().saveXML("RoomData.xml");
```

Wenn Sie jetzt die Datei „RoomData.xml“ einsehen, werden Sie feststellen, dass auch der Kundenname vorhanden ist.

```
<Data>
   <Rooms systemID="1" userID="" op="insert">
      <no_of_room>3</no_of_room>
      <start>2012-05-02T18:00:00.926</start>
      <end>2012-05-08T12:30:00.926</end>
      <amount>65.5</amount>
      <name_of_customer>Andreas Mustermann</name_of_customer>
    </Rooms>
  </Data>
```

Dieses einfache Beispiel zeigt Ihnen, wie Sie interaktive Datenerfassung beim Anlegen von Aktivitäten einbringen. Das einfache Beispiel ist natürlich entsprechend der Wünsche des Anwenders erweiterbar.

5 Die Gestaltung des Gantt-Diagramms

Nachdem Sie nun die grundlegenden Dinge der Erstellung eines Gantt-Diagramms und seine Veränderbarkeit verstanden haben, werden wir in diesem Kapitel verschiedene Möglichkeiten der optischen und funktionalen Gestaltung eines Gantt-Diagramms behandeln. Dabei werden wir nicht mehr den gesamten Programmiercode darstellen, sondern nur Code-Zeilen, die der Erreichung des gesteckten Zieles dienen. Von den Elementen Tabelle, Zeitskala, Gantt-Graph und Histogramm wenden wir uns zunächst der Tabelle zu.

5.1 Die Gestaltung der Tabelle

5.1.1 Aufbau und logische Struktur

Die Tabelle besteht aus Zeilen und Spalten.

	Ship	Worker	Task
1	⊟ Contest 34		
2	⊟ Contest 34	Davidson	
3	Contest 34	Davidson	Cabin
4	Contest 34	Davidson	Engine
5	⊟ Contest 34	Stewart	
6	Contest 34	Stewart	Cabin
7	Contest 34	Stewart	Engine
8	Contest 34	Stewart	Antifouling

	Ship	Worker	Task
1	⊟ Contest 34		
2	⊟ Contest 34	Davidson	
3	Contest 34	Davidson	Cabin
4	Contest 34	Davidson	Engine
5	⊟ Contest 34	Stewart	
6	Contest 34	Stewart	Cabin
7	Contest 34	Stewart	Engine
8	Contest 34	Stewart	Antifouling

Die oberste Zeile enthält die Spaltentitel, die erste Spalte enthält die Zeilentitel.

	Ship	Worker	Task
1	⊟ Contest 34		
2	⊟ Contest 34	Davidson	
3	Contest 34	Davidson	Cabin
4	Contest 34	Davidson	Engine
5	⊟ Contest 34	Stewart	
6	Contest 34	Stewart	Cabin
7	Contest 34	Stewart	Engine
8	Contest 34	Stewart	Antifouling

	Ship	Worker	Task
1	⊟ Contest 34		
2	⊟ Contest 34	Davidson	
3	Contest 34	Davidson	Cabin
4	Contest 34	Davidson	Engine
5	⊟ Contest 34	Stewart	
6	Contest 34	Stewart	Cabin
7	Contest 34	Stewart	Engine
8	Contest 34	Stewart	Antifouling

Der eigentliche Tabellenteil ohne die Zeilen- und Spaltentitel heißt auch Tabellenkern:

	Ship	Worker	Task
1	⊟ Contest 34		
2	⊟ Contest 34	Davidson	
3	Contest 34	Davidson	Cabin
4	Contest 34	Davidson	Engine
5	⊟ Contest 34	Stewart	
6	Contest 34	Stewart	Cabin
7	Contest 34	Stewart	Engine
8	Contest 34	Stewart	Antifouling

In jeder Tabellenzeile können Sie die Attributwerte einer Entität anzeigen. Den Zeileninhalt können Sie durch Zeilendefinitionen beschreiben. Dabei bilden mehrere gleichartige Zeilen einen Zeilentyp und benutzen dieselbe Zeilendefinition. Jede Zeilendefinition ist mit einer Priorität versehen und kann einen Filter beinhalten. Welche Zeilendefinition bei der Darstellung einer Zeile angewandt wird, hängt von der Priorität und vom Ergebnis der Filterauswertung ab. Eine Tabellenzeile besteht aus mehreren Feldern, wobei sich ein Feld über mehrere Tabellenspalten erstrecken kann. Deshalb setzt sich jede Zeilendefinition aus mehreren Felddefinitionen zusammen.

Den Inhalt und das Aussehen der Tabelle können Sie sowohl über bestimmte Eigenschaften von JGantt als auch über ein spezielles Objekt, die so genannte Inhaltsdefinition (contents definition), spezifizieren. Beide Verfahren verstehen sich nicht als gleichwertige Alternativen, sondern unterscheiden sich im Umfang ihrer Möglichkeiten zur Gestaltung der Tabelle und sollten komplementär verwendet werden.

5.1.2 Gestaltung über Eigenschaften

Die besonderen Merkmale bei der Tabellengestaltung über die Eigenschaften sind:

- Sie können nicht nur den Tabellenkern definieren, sondern auch das Aussehen der Spalten- und Zeilentitel.
- Jede Tabellenzeile hat die gleiche Anzahl von Feldern, d.h. die Tabelle hat durchgehende Spalten.

Es wird nur zwischen zwei Zeilentypen unterschieden, für die Sie separat die Eigenschaften einstellen können. Einen Typ bilden die Gruppenzeilen, in de-

nen die Gruppenknoten dargestellt werden (zum Thema "Gruppierung von Knoten" lesen Sie bitte das gleichnamige Kapitel weiter unten im Abschnitt über den Gantt-Graphen). Der andere Typ umfasst alle übrigen Zeilen, in denen die Basisknoten angezeigt werden.

	Ship	Worker	Task
1	⊟ Contest 34		
2	⊟ Contest 34	Davidson	
3	Contest 34	Davidson	Cabin
4	Contest 34	Davidson	Engine
5	⊟ Contest 34	Stewart	
6	Contest 34	Stewart	Cabin
7	Contest 34	Stewart	Engine
8	Contest 34	Stewart	Antifouling

Tabelle mit rötlichen Gruppenzeilen und gelblichen Basiszeilen

Die Eigenschaften können nur für eine ganze Zeile, nicht aber feldweise gesetzt werden. Sie können für jede Spalte nur einen Attributnamen angeben. Dies bedeutet, dass in einer Spalte nur die Werte von Attributen gleichen Namens angezeigt werden können. Existiert ein Attributname in einer der Entitätenmengen für die Knoten bzw. die Gruppenknoten nicht, dann bleiben die entsprechenden Felder in der Spalte leer. Die Eigenschaften können Sie zur Designzeit über die Eigenschaften-Tabelle von JGantt oder zur Laufzeit über die Programmierschnittstelle setzen. Die untenstehende Darstellung gibt einen Überblick über die Tabelleneigenschaften:

	group	start	end	duration
1	⊟ 1	14.01.2001	27.01.2001	10
2	⊟ 1.1	17.01.2001	25.02.2001	10
3	1.1	02.01.2001	10.02.2001	5
4	1.2	10.02.2001	25.02.2001	5
5	⊟ 2	10.02.2001	24.02.2001	10
6	2.1	18.01.2001	25.02.2001	10
7	2.2	02.01.2001	10.02.2001	5
8	3	02.01.2001	18.01.2001	5

tableColorScheme	banana
tableColumns	wbs<NL>description<NL>
tableColumnTitleFont	
tableColumnTitlesSource	entity attribute display names
tableColumnWidths	15.0<NL>15.0<NL>
tableDateFormat	
tableEditable	False
tableFont	"Arial", 0, 8
tableGroupFont	
tableHierarchyColumn	1
tableHierarchyIndentWidth	3.0
tableRowTitleFont	
tableRowTitlesVisible	True
tableVisible	True

Hier können Sie folgende Größen einstellen (Reihenfolge analog zu der in der Abbildung):

tableColorScheme - die Farbe für Zeilentitel, normale Knotenzeilen, Spaltentitel und Gruppenzeilen

tableColumns - die Datenquelle (Attributnamen der Knoten- bzw. Gruppenknotenentitäten) für jede Spalte. Hierdurch wird gleichzeitig die Anzahl der Tabellenspalten festgelegt.

tableColumnTitleFont - die Schriftart der Spaltentitel

tableColumnTitlesSource - hier legen Sie fest, ob die Spaltentitel aus den Attributnamen oder aus den Display-Namen (siehe Eigenschaft **Entity-AttributeDisplayName** in der Schnittstelle **NeIEntitySet**) gebildet werden.

tableColumnWidth - die Breite der Spalten

tableDateFormat - das Datumsformat

tableEditable - die Editierbarkeit der Tabelle

tableFont - die Schriftart für die Felder der Basiszeilen

tableGroupFont - die Schriftart für die Felder der Gruppenzeilen

tableHierarchyColumn – hier geht es um hierarchische Darstellungen, die in einem späteren Kapitel detailliert behandelt werden. Die Eigenschaft gibt die Nummer der Spalte an, in der die Felder entsprechend der hierarchischen Struktur eingerückt dargestellt werden sollen. Sie ist nur relevant, wenn die Eigenschaft **groupMode** auf **JGantt.GROUP_BY_HIERARCHY** gesetzt ist.

tableHierarchyIndentWidth - die Größe der Einrückung pro Ebene von den Feldinhalten in der Spalte, die zur Darstellung der Hierarchie ausgewählt worden ist.

tableRowTitleFont - die Schriftart der Zeilentitel

tableRowTitlesVisible - die Sichtbarkeit der Spaltentitel

tableVisible - die Sichtbarkeit der gesamten Tabelle

5.1.3 Gestaltung über die Inhaltsdefinition

Die folgenden Punkte geben einen groben Überblick über die Gestaltungsmöglichkeiten mittels Inhaltsdefinition: Die Inhaltsdefinition stellt nur erweiterte Möglichkeiten zur Gestaltung des Tabellenkerns bereit. Zeilen- oder Spaltentitel können Sie derzeit damit nicht ansprechen. Sie können auch Felder definieren, die sich über mehrere Spalten hinweg erstrecken. Es lassen sich beliebig viele Zeilentypen definieren.Bei jedem Zeilentyp können Sie das Aussehen der Felder einzeln definieren. In jeder Spalte können von Zei-

lentyp zu Zeilentyp die Werte unterschiedlicher Attribute dargestellt werden. Sie haben den vollen Zugriff auf die Inhaltsdefinition nur über die Programmierschnittstelle. Auf die Inhaltsdefinition **NeContentsDefinition** erhalten Sie Zugriff über die Komponente Table:

```
NeContentsDefinition contDef;
…
contDef = jGantt1.getTable().getContentsDefinition();
```

In jeder Zeile der Tabelle können Attributwerte einer Entität angezeigt werden. Den Zeileninhalt beschreiben Sie durch Zeilendefinitionen **NeRowDefinition**. Dabei bilden mehrere gleichartige Zeilen einen Zeilentyp und benutzen dieselbe Zeilendefinition. Ein Objekt der Klasse **NeContentsDefinition** enthält eine Menge solcher Zeilendefinitionen. Dabei besitzt es auf jeden Fall immer mindestens eine Zeilendefinition, auf die über die Eigenschaft **DefaultRowDefinition** zugegriffen werden kann. Jede Zeilendefinition besteht aus einer Reihe von Felddefinitionen **NeFieldDefinition**, in denen festgehalten ist, welcher Inhalt (z.B. welches Entitäten-Attribut) im zugehörigen Feld dargestellt werden soll.

```
NeContentsDefinition contDef;
Iterator iter;
NeRowDefinition defaultRowDef;
…
contDef       = jGantt1.getTable().getContentsDefinition();
defaultRowDef = contDef.getDefaultRowDefinition();

// Hinzufügen zweier Spalten über zwei weitere fieldDefinitions
// zur default row definition.

// Erste Spalte zeigt Werte des Entitäten-Attribus "taskName"
defaultRowDef.addFieldDefinition(new NeFieldDefinition("taskName")

// Zweite Spalte zeigt Werte des Entitäten-Attribus "startDate"
defaultRowDef.addFieldDefinition(new NeFieldDefinition("startDate")

// Iteratorobjekt anlegen um die field definitions zu durchlaufen
iter = defaultRowDef.iterateFieldDefinitions();
```

Die Standardzeilendefinition bestimmt auch die Anzahl der Tabellenspalten. Die Tabelle enthält genauso viele Spalten, wie die Standardzeilendefinition Felder hat. Damit stets eine Tabellendarstellung sichergestellt ist, können Sie die Standardzeilendefinition nicht löschen.

Zu der Standardzeilendefinition können Sie beliebig viele weitere Zeilendefinitionen hinzufügen. Dabei ist jede Zeilendefinition mit einer Priorität verse-

hen und kann zudem einen Filter beinhalten. Welche Zeilendefinition bei der Darstellung einer Zeile angewandt wird, hängt von der Priorität und vom Ergebnis der Filterauswertung ab. Der Standardzeilendefinition ist kein Filter zugeordnet, und deshalb wird sie immer dann angewendet, wenn keine andere Zeilendefinition in Frage kommt.

```
// Neue Zeilendefinition wird der Inhaltsdefinition hinzugefügt

NeRowDefinition      newRowDef;
NeIFilter            filter;
NeContentsDefinition contDef = jGantt1.getTable()
                                      .getContentsDefinition);

// Anlage eines Filters, um "critical" Entitäten herauszufiltern

filter = new NeFilterBase ()
   {
   public boolean apply (NeIEntity entity)
      {
       return (entity.getValueAsBoolean("critical") == true);
      }
   };

// Anlage einer Zeilendefinition mit Filter und Priorität 2
newRowDef = new NeRowDefinition(filter, 2);

// Zeilendefinition der Inhaltsdefinition hinzufügen
contDef.addRowDefinition(newRowDef);
```

Das Aussehen der Felder (z.B. Feldhintergrundfarbe oder Schriftart) wird über Feldstile **NeFieldStyle** definiert. Dabei gibt es zwei Möglichkeiten, Feldstile einzusetzen: Entweder weisen Sie einer Zeilendefinition einen Stil zu. Dann sehen alle Felder eines Zeilentyps gleich aus. Oder Sie weisen einer Felddefinition einen Stil zu, wenn Sie einzelnen Feldern eines Zeilentyps ein unterschiedliches Aussehen geben wollen. Dieser Feldstil wird immer dann zum Zeichnen eines Feldes verwendet, wenn einer Felddefinition kein eigener Feldstil zugewiesen wurde.

```
NeFieldStyle fieldStyle;
NeRowDefinition rowDef = jGantt1.getTable().getContentsDefinition()
                                           .getDefaultRowDefinition();

// Alle Felder der default row definition haben denselben fieldStyle,
// Feldhintergrund ist rot, Schriftfarbe blau

fieldStyle = new NeFieldStyle(new Font("Helvetica", Font.PLAIN, 8),
    NeIDrawingConstants.ALIGNMENT_CENTER_LEFT,Color.blue, Color.red);
rowDef.setFieldStyle(fieldStyle);

// Erstes Feld erhält eigenen fieldStyle mit schwarzem Text und grünem
// Hintergrund
```

```
fieldStyle = new NeFieldStyle(new Font("Helvetica", Font.PLAIN, 10),
    NeIDrawingConstants.ALIGNMENT_CENTER_LEFT,Color.black, Color.green);
rowDef.getFieldDefinition(0).setFieldStyle(fieldStyle);
```

Ein Feld in einer Tabelle entspricht in der Regel der Kreuzungsfläche zwischen einer Zeile und einer Spalte. Allerdings kann einem Feld auch eine Spannweite (range) über mehrere Spalten zugewiesen werden. Dies ist allerdings nicht bei der Standardzeilendefinition möglich, denn diese wird ja - wie oben bereits erwähnt – zum Festlegen der Anzahl der Tabellenspalten verwendet. Die Spannweite wird nicht bei der Felddefinition, sondern bei der Zeilendefinition eingestellt. Das hat den Vorteil, dass Sie dieselbe Felddefinition mehrfach, auch in mehreren Zeilendefinitionen mit unterschiedlichen Spannweiten, verwenden können.

```
// Die Tabelle unten habe 10 Spalten, von denen jeweils 5 verbunden
// sein sollen.
// Die Zeilendefinition enthält daher zwei Felddefinitionen, die sich
// über jeweils 5 Spalten erstrecken

NeRowDefinition   rowDef;
NeFieldDefinition fieldDef1, fieldDef2;

rowDef.setFieldRange(fieldDef1, 5)
rowDef.setFieldRange(fieldDef2, 5)
```

5.1.4 Weitere Tabellenfunktionalität

Neben dem Grundaufbau gibt es eine Methode, mit der Sie die optimale Breite einer Tabellenspalte berechnen können. Optimale Spaltenbreite bedeutet, dass die Texte in den Zellen dieser Spalte vollständig lesbar sind, und gleichzeitig ist die Spalte nur so breit wie notwendig. Die Methode liefert die eingestellte Breite in Millimeter zurück:

```
double calcOptimizedTableColumnWidth (int columnIndex)
```

In dem untenstehenden Programmierbeispiel wird die optimale Breite der Spalte 2 ermittelt und eingestellt. Anschließend wird diese Breite auf die benachbarten Spalten übertragen, damit alle Spalten gleich aussehen.

```
double optimumColumnWidth;

// Optimale Breite der Spalte 2 ermitteln
optimumColumnWidth = jGantt1.calcOptimizedTableColumnWidth(2);

// Übertragung dieser Breite auf alle anderen Spalten
jGantt1.setTableColumnWidths(0, optimumColumnWidth);
```

```
jGantt1.setTableColumnWidths(1, optimumColumnWidth);
jGantt1.setTableColumnWidths(3, optimumColumnWidth);
jGantt1.setTableColumnWidths(4, optimumColumnWidth);
```

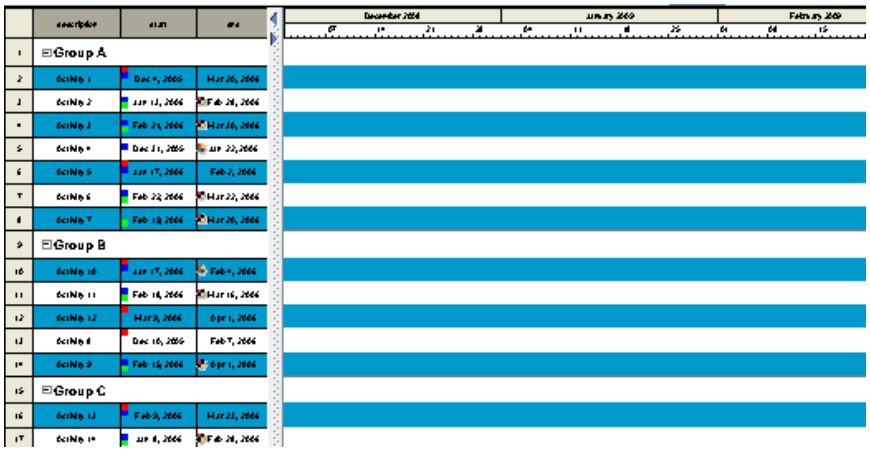
Wenn nach dem Einstellen der optimalen Spaltenbreite sich die Texte in der Spalte verändern, dann wird nicht automatisch die Spaltenbreite auf das neue Optimum eingestellt. Dazu muss dann der Befehl **calcOptimizedColumn-Width** erneut aufgerufen werden. Alternativ zur Einstellung über die Programmierschnittstelle kann der Benutzer die optimale Spaltenbreite auch interaktiv dadurch setzen, dass er auf den Titel der entsprechenden Spalte doppelt klickt.

Mit dem Abschluss dieses Kapitels haben Sie die Grundlagen zur Gestaltung der Tabelle erworben. Wir werden uns jetzt der Gestaltung des Gantt-Graphen zuwenden.

5.2 Zeilenfarben für Tabelle und Gantt-Graph

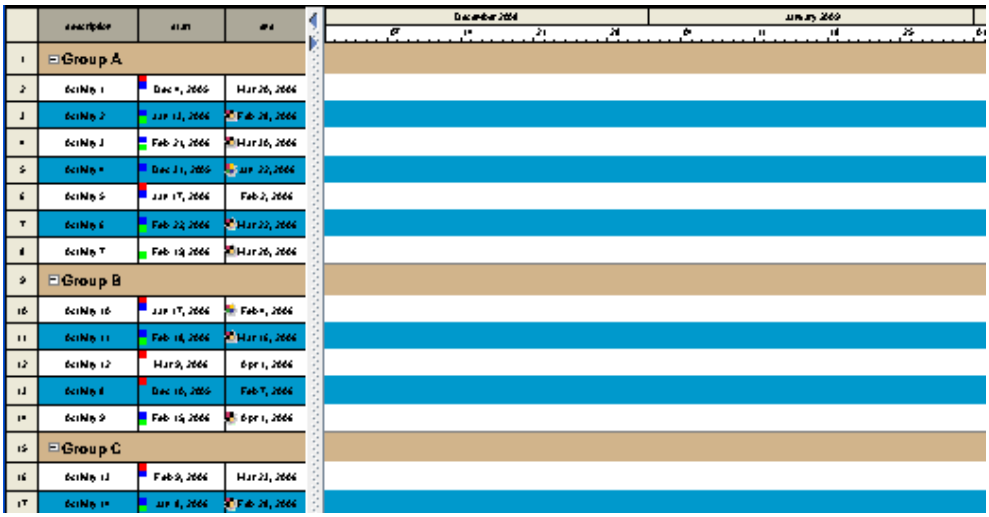
Sie können die Tabelle wie auch den Gantt-Graphen zeilenweise einfärben. Dafür stehen Ihnen die Konstruktoren der Klasse **JGDynamicRowColor** zur Verfügung. Mit ihnen können Sie die Zeilen von Tabelle und Gantt-Graph z.B. wechselweise einfärben. Wir zeigen Ihnen im Beispiel unten, wie Sie eine Farb-Variable, die in unserem Beispiel dynamicRowColor heißt, über den Konstruktor mit alternierenden Farben belegen:

```
Color dynamicRowColor = new JGDynamicRowColor (jGantt,Color.white,
                                                new Color (0,153,204));
```



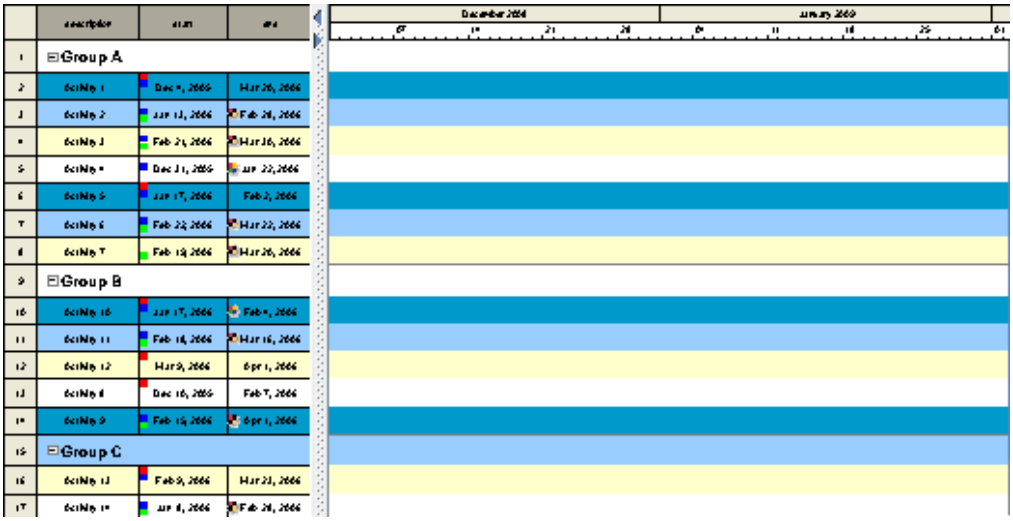
Wenn Sie einen zusätzlichen Gruppenparameter setzen (der Begriff „Gruppe“ wurde bisher noch nicht erläutert, aber behandeln Sie ihn zunächst als „Blackbox“, deren Inhalt sich in den kommenden Kapiteln noch erschließen wird), werden Gruppen mit einer eigenen Farbe versehen:

```
Color dynamicRowColor = new JGDynamicRowColor (jGantt,
                        Color.white,
                        new Color (0,153,204),
                        new Color (209, 180, 139));
```



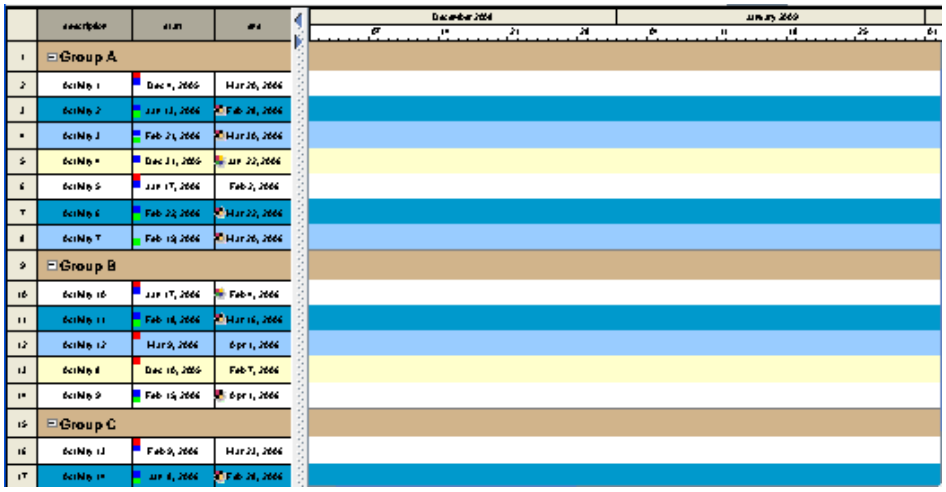
Sie können über einen weiteren Konstruktor von **JGDynamicRowColor** auch einen Array von Farben definieren, der sich rotierend wiederholt:

```
Color dynamicRowColor = new JGDynamicRowColor (jGantt,
                                    new Color [] {Color.white,
                                    new Color (0, 153, 204),
                                    new Color (153, 204, 255),
                                    new Color (255, 255, 204)}, false);
```



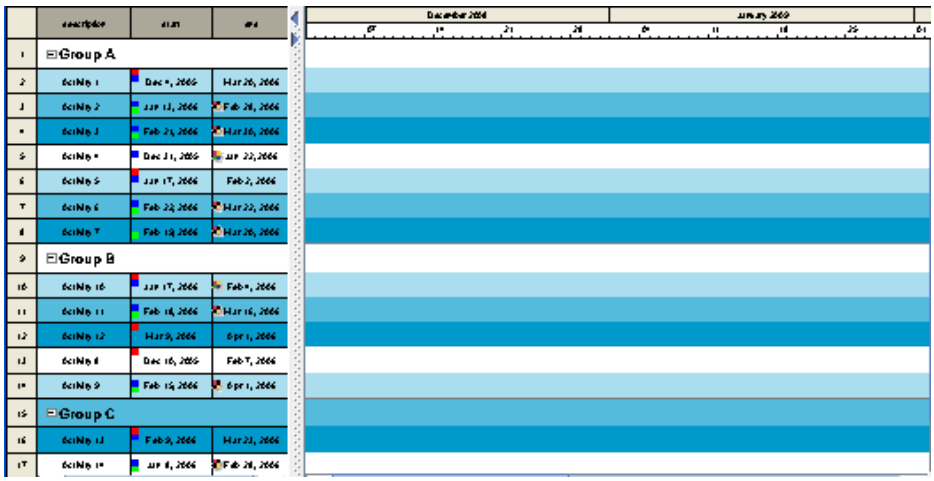
Wenn Sie bei diesem Konstruktor den Gruppenparameter auf **true** setzen, beginnt der Array für jede Gruppe, von vorne zu zählen. Dabei werden höchstens n-1 Farben für die Gruppe verwendet; Farbe n ist für die Zeile mit dem Gruppenkonten reserviert:

```
Color dynamicRowColor = new JGDynamicRowColor (jGantt,
                                new Color [] {Color.white,
                                new Color (0, 153, 204),
                                new Color (153, 204, 255),
                                new Color (255, 255, 204),
                                new Color (209, 180, 139)},true);
```



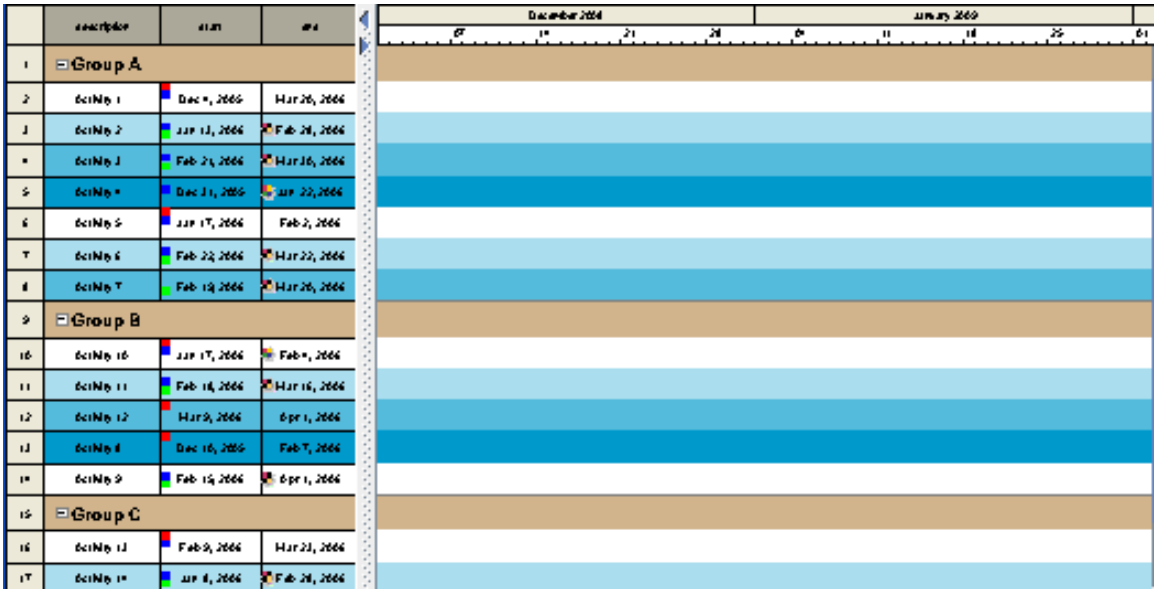
Sie können auch die Diagrammzeilen als Farbverläufe darstellen. Dabei geben Sie die beiden Grenzfarben an sowie die Anzahl der Farbabstufungen, über die sich der Farbverlauf erstrecken soll:

```
Color dynamicRowColor = new JGDynamicRowColor (jGantt,
                            new Color (170,221,238),
                            new Color (0, 153, 204), 4, null);
```



Wenn Sie noch eine zusätzliche Gruppenfarbe angeben, können Sie für Gruppen eine eigenen Zeilenfarbe einrichten:

```
Color dynamicRowColor = new JGDynamicRowColor (jGantt,
                        new Color (170,221,238),
                        new Color (0, 153, 204), 4,
                        new Color (209, 180, 139));
```



Nachdem Sie die Werte in den Konstruktoren so gesetzt haben, wird nun die Variable **dynamicRowColor** den Zeilen im Farbschema des Gantt-Graphen zugewiesen:

```
colorScheme = jGantt.getGanttColorScheme ();
colorScheme.setShadedColor (dynamicRowColor);
jGantt.setGanttColorScheme (colorScheme);
```

…und der Standard-Tabellenzeile:

```
JGColorScheme colorScheme = jGantt.getTableColorScheme ();
colorScheme.setShadedColor (dynamicRowColor);
jGantt.setTableColorScheme (colorScheme);
```

Wenn Sie Tabellenzeilen über eine Inhaltsdefinition gestaltet haben, weisen Sie die Farbe dem dort verwendeten FieldStyle-Objekt zu:

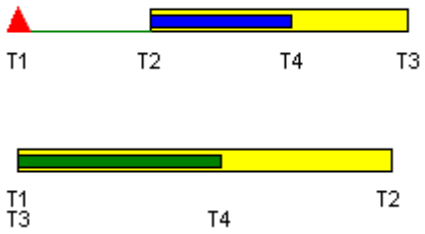
```
NeFieldStyle groupNameStyle = new NeFieldStyle (new Font ("Arial",
                                  Font.BOLD, 12),
                                  NeIDrawingCon-
stants.ALIGNMENT_CENTER_LEFT,
                                  Color.black, dynamicRowColor);
```

Damit haben Sie sich nun die Grundlagen für die farbige Gestaltung von Zeilen in Tabelle und Gantt-Graph erarbeitet. Wir werden uns nun weiter der Gestaltung des Gantt-Graphen zuwenden.

5.3 Knoten im Gantt-Graphen

5.3.1 Knotentermine

Knoten sind die zentralen Elemente des Gantt-Graphen. Sie haben bereits beim Verändern eines bestehenden Programms feststellen können, wie groß die Auswahl an verschiedenen Knotendesigns ist. Unter ihnen gibt es auch solche, die Fortschrittsbalken zeigen, wie z.B. das Design **LineCompletion-Rect** oder **2RectsInterlocked**.



Bei ihnen kann es sich als notwendig erweisen, die Länge des innenliegenden Balkens als prozentualen Anteil des außenliegenden Balkens anzugeben. Wir zeigen Ihnen in diesem Kapitel, welche Möglichkeiten VARCHART JGantt bietet, Termine zu definieren.

Grundsätzlich können Sie einen Termin als absoluten oder als relativen Termin angeben. Bei absoluten Terminen können Sie das Datum direkt über einen Attributnamen vorgeben, z.B.:

```
jGantt1.setNodeDates(new String[]{"T1", "T2", "T3", "T4"});
```

Sie können bei einem absoluten Termin auch noch eine Dauer addieren, z.B.

```
jGantt1.setNodeDates(new String[]{"T1","T1+duration(d)", "T3", "T4"});
```

Die Dauer wird aus dem Attribut mit dem Namen **duration** entnommen und durch den in Klammern angegebenen Buchstaben, in unserem Beispiel d, mit einer Einheit versehen. Zugelassen sind als Einheiten d oder D für Tag, h oder H für Stunde, m oder M für Minute und s oder S für Sekunde. Falls Sie keine Einheit angeben, wird die angegebene Dauer automatisch in Millisekunden interpretiert. Bei einem relativen Termin kann der Anteil einer Dauer berechnet werden, die zwischen zwei absoluten Terminen liegt. Der Faktor, der den Anteil bestimmt, steht in eckigen Klammern und muss als Attributname angegeben werden:

```
"T1[percentage]T2"
```

Wenn das Attribut **percentage** den Wert 0,5 enthält, dann liegt dieser Termin auf halber Strecke zwischen T1 und T2. Sie können auch die Einheit Prozent in der Formel angeben. Dann wird der Inhalt des Attributs in Prozent umgerechnet, und der Wert müsste 50 betragen, wenn dasselbe Ergebnis wie im vorangegangenen Beispiel erreicht werden soll:

```
"T1[percentage(%)]T2"
```

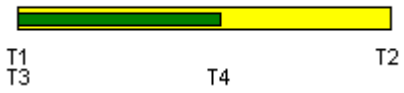
In den obigen Beispielen haben Sie einen relativen Termin als Anteil einer Dauer definiert, die durch zwei Termine festgelegt wurde. Alternativ können Sie einen relativen Termin definieren, bei dem Sie nur auf einen einzigen absoluten Termin zurückgreifen:

```
"T1[percentage]T1+duration(d)"
```

Wenn T1 den Wert 1.1.2010 enthält, **percentage** den Wert 0,5 und **duration** den Wert 2, dann werden 0,5*2 Tage zum 1.1.2010 addiert und der definierte Termin liegt auf dem 2.1.2010. Als Abschlussübung reduzieren wir nun die vier absoluten Termine

```
jGantt1.setNodeDates(new String[]{"T1", "T2", "T3", "T4"});
```

die das Design **2RectsInterlocked** auf die unten abgebildete Weise



darstellen sollen, auf ein einziges absolutes Terminattribut:

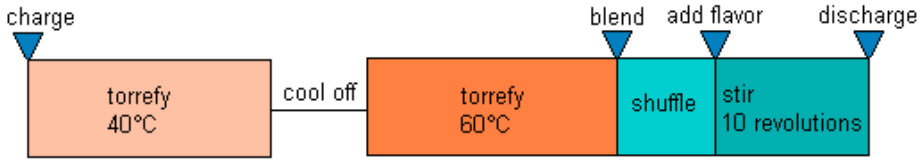
```
jGantt1.setNodeDates(new String[]
    {"T1", "T1+duration(d)", "T1", "T1[percentage]T1+duration(d)"});
```

Sie haben nun einfache und komplexe Formen der Termindefinition kennengelernt, die der Darstellung einzelner Knoten dienen. Wir werden in den folgenden Kapiteln weniger den Einzelknoten, sondern eher die Gesamtheit der Knoten und ihre Anordnung im Gantt-Graphen betrachten.

5.3.2 Gestaltung von Knoten

Sie haben bereits erfahren, wie Sie Knoten über vordefinierte Formate zusammensetzen. Wir zeigen Ihnen hier, wie Sie Knoten über die Programmierschnittstelle völlig frei gestalten können. Die graphische Struktur von Knoten besteht aus graphischen Elementen zweier Kategorien. Da gibt es zum einen die Layer, die meist einen Balken oder eine Linie darstellen:

Der obige Knoten z.B. besteht aus fünf Layern: einem Linien-Layer und vier Balken-Layern.

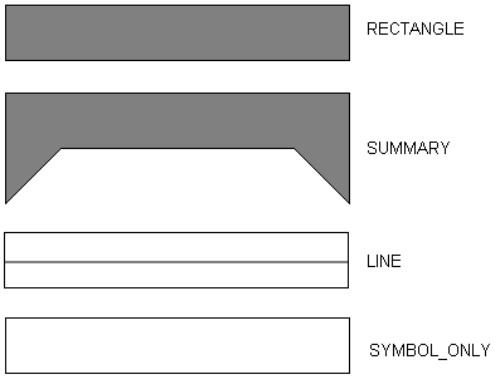


Die Layer können mit Dekorationen ausgestattet werden, sie bilden die zweite Kategorie von graphischen Elementen und bestehen entweder aus Symbolen, Bitmaps oder Beschriftungen. Ein Layer wird beschrieben von einer Layer-Definition, die Sie über die Methode **createLayerDefinition()** anlegen. Ein derartig angelegter Layer ist automatisch bei allen Knoten des JGantt-Objektes präsent. Sie können seine Präsenz jedoch über einen Filter verhindern, den Sie der Layer-Definition zuweisen können.

Grundsätzlich ist ein Layer ein Rechteck. Es hat eine Länge, die durch eine Dauer definiert ist. Diese Dauer ist durch ein Anfangs- und Enddatum begrenzt. Anfangs- und Enddatum können identisch sein, dann hat der Layer keine horizontale Ausdehnung. Anfangs- und Endtermin eines Layers legen Sie über die Eigenschaften **StartValueReference** und **EndValueReference** der Layerdefinition fest.

ValueReference-Objekte implementieren das Interface **NeIValueReference** und repräsentieren das Bindeglied zwischen den Layer-Terminen und den in der Entität eines Knotens vorliegenden Terminattributen für Beginn und Ende. Die Höhenausdehnung eines Layer-Rechtecks legen Sie durch die Eigenschaft **Height** fest. Weiterhin wird die relative Lage zur Knotenmitte über die Eigenschaft **VerticalOffset** definiert. Wie ein Layer dargestellt wird, legen

Sie über den Layer-Typ fest. Der Typ **RECTANGLE** füllt das definierte Rechteck aus. Der Typ **SUMMARY** füllt das Rechteck aus und setzt an seine Unterseite Dreiecke von gleicher Höhe. Der Typ **LINE** setzt eine Linie in das Rechteck. Bei dem Typ **SYMBOL_ONLY** erfolgt überhaupt keine Darstellung des Layers, doch können Sie dekorative Elemente zuweisen.



Dekorative Elemente, die zweite Kategorie von graphischen Elementen, werden an die Layer gesetzt und bestehen aus Bitmaps, Symbolen oder Beschriftungen. Es gibt hinsichtlich des Ausdehnungsverhaltens zwei Arten von Dekorationen: **Labels** und **Pictures**.

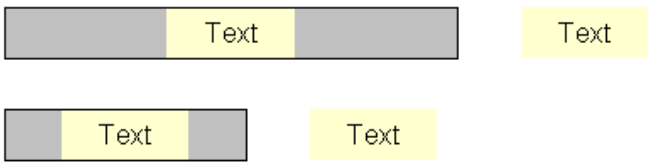
Im Gegensatz zu Pictures werden Labels einem bestimmten Punkt des Layers zugewiesen. Aussehen und Größe hängen von der implementierenden Klasse ab und werden durch die Eigenschaft **Extent** festgelegt.

Pictures hingegen werden in Bereiche hineingezeichnet, deren Größe vom Layer vorgegeben wird. Der Bereich kann im Layer selbst liegen oder außerhalb, z.B. oberhalb oder unterhalb des Layers. Wenn das als Picture gezeichnete Dekorationsobjekt innerhalb des Layers liegt, kann es den Layer ganz oder teilweise überdecken, sofern es optisch dicht ist. Wenn es durchsichtig ist, bleibt der Layer darunter sichtbar. Auch einem als Picture übergebenen Dekorationsobjekt können Sie eine Größe zuweisen. Sie wird als bevorzugte Größe behandelt und wird vom Layer bei Berechnungen so weit als möglich berücksichtigt.

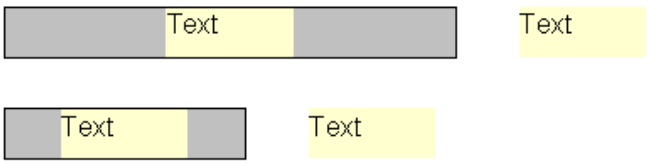
Falls Sie das Ausdehnungsverhalten automatisieren (Standardverhalten), bewirkt sie, dass das Objekt möglichst in der bevorzugten Größe so häufig wiederholt wird, bis es die vom Layer vorgegebene Fläche ausfüllt. Dabei werden nur ganze Objekte gezeichnet. Weisen Sie keine Größe zu, wird das Dekorationsobjekt auf die vom Layer vorgegebene Größe gedehnt. Falls Sie das Ausdehnungsverhalten nicht automatisieren, bieten sich zahlreiche Varianten manueller Einstellung.

Für die Definition von Dekorationen stellt VARCHART JGantt drei Klassen zur Verfügung: **NeAnnotation** für Knotenbeschriftungen, **NeSymbol** für vektorbasierte Symbole und **Bitmap** für Bitmaps. Alle Klassen implementieren sowohl **NeIPicture** als auch **NeILabel**. Den Symbolen und Bitmaps können Sie verschiedene Styles für Umrandungen und Fläche zuweisen; weiterhin können Sie durch die Veränderung der Eigenschaften **HorizontalPictureFillMode**, **VerticalPictureFillMode**, **Extent**, **StretchMode** und **FontSize** zahlreiche Variationen in der Darstellung erreichen. Wir stellen im Folgenden ein paar typische Möglichkeiten vor.

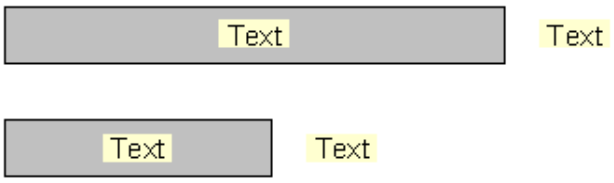
Grundlegend für das Verstehen des Verhaltens von Symbolen, Beschriftungen und Bitmaps ist, dass diese Darstellungen aus einem Vordergrund und Hintergrund bestehen, deren Verhalten sich bei den beiden letzteren getrennt einstellen lässt. Der Hintergrund von Beschriftungen heißt Beschriftungsrechteck. Im Bild unten sehen Sie vier Beschriftungen; davon die beiden linken implementiert als **NeIPicture** in je einem Layer (schwarzer Rahmen) und die beiden rechten außerhalb, implementiert als **NeILabel**. Das gelbe Beschriftungsrechteck befindet sich unmittelbar hinter dem Text und überragt diesen nach allen Seiten:



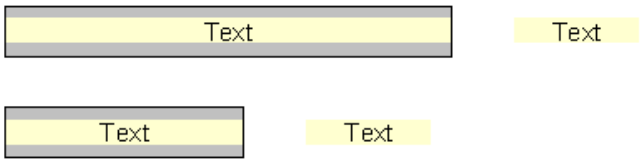
Sie können das Aussehen der Schrift über NeAnnotation.ScaleFontToFit verändern und über **NeAnnotation.ClipMode** oder **NeAnnotation.AutoWrapMode** das Umbruchverhalten des Textes steuern. Außerdem können Sie den Text innerhalb des Beschriftungsrechtecks positionieren:



Im Bild oben wurde über **NeAnnotation.Alignment** der Text in die linke obere Ecke des Beschriftungsrechtecks positioniert. Auch die Größe des Beschriftungsrechtecks kann geändert werden. Hier die halbierte Größe, eingestellt über **NeAnnotation.Extent**:



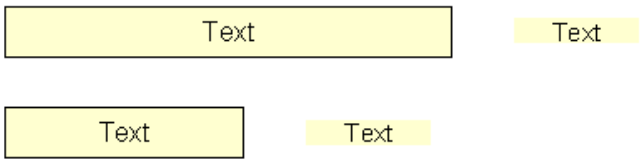
Für die grafische Gesamtgestaltung des Layers ist nun die Positionierung des gesamten Beschriftungsrechtecks relevant. Sie können dazu das Füllverhalten des Beschriftungsrechtecks für den Layer einstellen. Dazu verwenden Sie die Eigenschaften **HorizontalPictureFillMode** und **VerticalPictureFillMode** der Klasse **NeAnnotation**. Diese Eigenschaften greifen nur, wenn die Beschriftung als **NeIPicture**-Objekt verwendet wird. Die Bilder oben zeigen für beide Richtungen die Einstellung **CENTER**. Das Beschriftungsrechteck wird dabei singulär in die horizontale und vertikale Mitte des Layers gesetzt. Entsprechendes gilt für **TOP**, **BOTTOM**, **LEFT** und **RIGHT**. Es gibt aber auch die Möglichkeit, das Beschriftungsrechteck zu dehnen oder es kachelartig zu wiederholen:



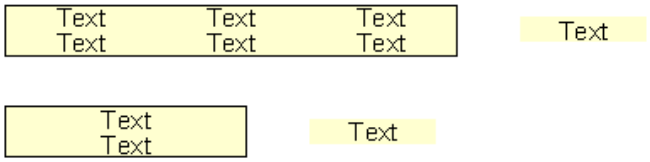
Im Bild oben wurde das Beschriftungsrechteck mittels

```
setPictureFillMode(NeAnnotation.STRETCH);
```

in horizontale Richtung gedehnt. Sie können es entsprechend auch vertikal dehnen, so dass es den Layer komplett ausfüllt:



Hier nun das Kacheln mit dem Beschriftungsrechteck:



Die beiden Füllmodi wurde jeweils auf "Kacheln" gesetzt:

```
setHorizontalPictureFillMode(NeAnnotation.TILE);
setVerticalPictureFillMode(NeAnnotation.TILE);
```

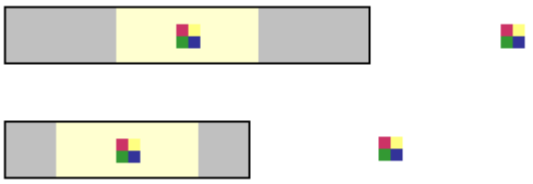
Wenn Sie die Füllmodi automatisieren:

```
setHorizontalPictureFillMode(NeAnnotation.AUTO);
setVerticalPictureFillMode(NeAnnotation.AUTO);
```

wird, wie bereits erwähnt, das Beschriftungsrechteck in beide Richtungen gekachelt, sofern über **Extent** eine Größe gesetzt wurde. Ohne vorgegebene Größenwerte wird es in die eine oder in beide Richtungen auf das Ausmaß des Layers gedehnt.

Wenn Sie mit einer Bitmap der Klasse **NePicture** arbeiten, können Sie zum einen festlegen, ob und wie die Bitmap vor ihrem Hintergrund gedehnt, gekachelt oder positioniert wird; zum andern können Sie den Hintergrund innerhalb des Layers dehnen, kacheln oder positionieren. Ersteres ist deshalb möglich, weil Vordergrund und Hintergrund bei einer Bitmap getrennt angesprochen werden können; für ein Symbol (Klasse NeSymbol) ist dies nicht möglich, da ein Symbol fest mit seinem Hintergrund verbunden ist. Bei einer Bitmap stellen Sie die Optionen des Vordergrundes auf dem Hintergrund über die Eigenschaften **StretchMode** und **Alignment**, die Optionen des Hintergrundes vor dem Layer über die Eigenschaften **HorizontalPictureFillMode** und **VerticalPictureFillMode** der Klasse **NePicture** ein. Im Bild unten sehen Sie vier Bitmaps, die sich in ihrer Original-Größe in der linken Bildhälfte vor ihrem gelben Hintergrund in zwei Layern befinden und rechts ohne Hintergrund außerhalb von Layern. Über die Eigenschaft **Extent** wurde die Größe des gelben Hintergrunds gesetzt:

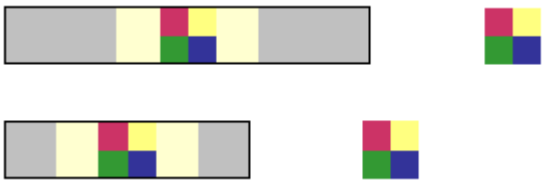
```
setExtent(1000,400);
setAlignment(NePicture.ALIGNMENT_CENTER_CENTER);
setHorizontalPictureFillMode(NePicture.CENTER);
setVerticalPictureFillMode(NePicture.AUTO);
setStretchMode(NePicture.NOT_STRETCHED);
```



Falls Sie mit Extent keinen Wert setzen, nimmt die Ausdehnung des Hintergrundes die "natürliche" Größe der Bitmap an. Diese ergibt sich aus Größe (in Pixel) * 2540 / 72.

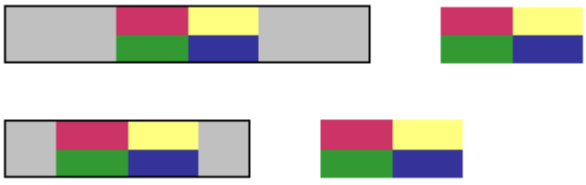
Im Bild oben sitzt der Vordergrund zentriert auf dem Hintergrund (Alignment). Der Hintergrund sitzt zentriert auf der Layer-Fläche (Füll-Modi). Der Vordergrund ist im Bild oben ungedehnt. Wenn Sie den Vordergrund nun dehnen, wird er soweit gedehnt, bis der Vordergrund den Hintergrund in mindestens einer Richtung ausfüllt. Wenn der Dehnungsmodus auf **STRETCHED** gesetzt wird, wird das Bild also nicht verzerrt:

```
setStretchMode(NePicture.STRETCHED);
```



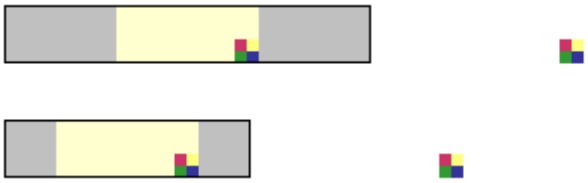
Dies ist anders bei den verbleibenden Dehnungsmodi. Sie können in die horizontale, vertikale oder, wie abgebildet, in beide Richtungen dehnen, so dass der Hintergrund in die jeweils genannte Richtung ausgefüllt wird:

```
setStretchMode(NePicture.STRETCHED_ANISOTROPIC);
```



Sie können nun, ähnlich wie bei einer Beschriftung, den Vordergrund auf dem Hintergrund des Picture-Objektes positionieren. Dies erfolgt entsprechend über die Eigenschaft Alignment. Um die Wirkung besser sehen zu können, nehmen wir die Dehnung zurück:

```
setStretchMode(NePicture.NOT_STRETCHED);
setAlignment(NePicture.ALIGNMENT_BOTTOM_RIGHT);
```

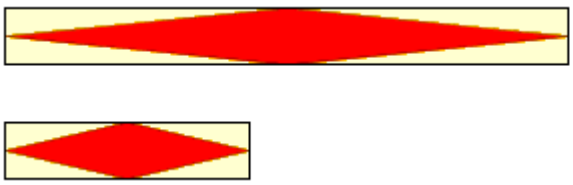


Im Bild oben wurde der Vordergrund in die rechte untere Ecke des Hintergrundes platziert.

Sie können – wie bei der Beschriftung - den Layer mit der Bitmap über ähnlich Optionen ausfüllen. Beispielhaft haben wir den Layer mit der obigen Einstellung horizontal gekachelt:

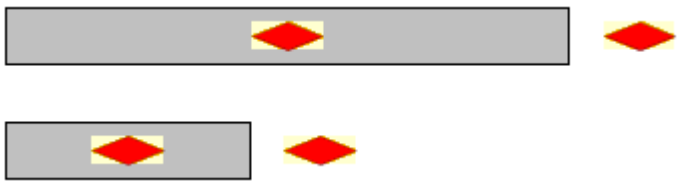
```
setHorizontalPictureFillMode(NePicture.TILE);
```

Anders als Beschriftungen und Picture-Objekte verhalten sich die Symbole. Bei ihnen sind Hintergrund und Vordergrund fest verbunden, wodurch die Darstellungsmöglichkeiten weniger variantenreich sind.Ein Symbol hat keine eigene Größe; aus diesem Grund wird es, falls sie über **NeSymbol.Extent** keine Größe zuweisen, stets auf die Größe des Layers gedehnt, unabhängig davon, welche Einstellungen Sie beim horizontalen und vertikalen Füllmodus vornehmen:

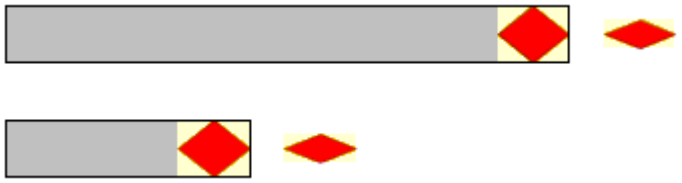


Im Bild oben sehen Sie nur zwei Symbole, die auf die vorgegebene Rahmengröße des Layers gedehnt wurden. Eine Darstellung der Symbole rechts da-

von (wie in den vorhergehenden Bildern) ist nicht möglich wegen der fehlenden eigenen Größe. Sobald Sie diese zuweisen, wird das Symbol auch außerhalb eines Layers darstellbar:



Hier wurde das Symbol über den horizontalen und vertikalen Füllmodus jeweils mittig positioniert; Sie können natürlich auch hier anders positionieren, dehnen oder kacheln:



```
setHorizontalPictureFillMode(NeSymbol.RIGHT);
setVerticalPictureFillMode(NeSymbol.STRETCH);
```

Oben wurde das Symbol an die rechte Seite des Layers geschoben und auf seine Höhe gedehnt, unten wurde der horizontale Füllmodus automatisiert und in der Vertikalen nach oben positioniert:

```
setHorizontalPictureFillMode(NeSymbol.AUTO);
setVerticalPictureFillMode(NeSymbol.TOP);
```

Das automatische Verhalten eines Symbols zeigt sich bei nicht angegebener Größe als Dehnen; bei vorgegebener Größe wird gekachelt.

Sie haben nun die grafischen Gestaltungsmöglichkeiten mit Dekorationsobjekten vertiefend kennen gelernt. Bevor wir nun zur Programmierung eines frei zusammensetzbaren Knotens fortschreiten, werden wir noch einen ge-

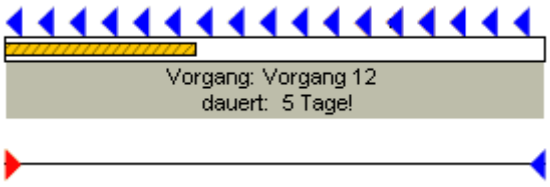
naueren Blick darauf werfen, wie Sie komfortabel die End- und Starttermine eines Layers verwalten.

ValueReference-Objekte

VARCHART JGantt stellt ValueReference-Objekte zur Verfügung, die Sie für die Verwaltung des End- und Starttermins eines Layers verwenden. Sie erlauben eine besonders komfortable Handhabung der Termine, insbesondere, wenn sie sich aus verschiedenen Anteilen zusammensetzen, wie z.B. einem Anfangstermin und einer Dauer. ValueReference-Objekte können Sie von drei verschiedenen Klassen ableiten: von **NeValueReference**, **NeSumValueReference** und **NeRelativeValueReference**. Zudem steht Ihnen das Interface **NeIValueReference** mit weiteren implementierten Eigenschaften zur Verfügung.

Die Klasse **NeValueReference** bietet Ihnen mit den Eigenschaften **name**, **factor** und **offset** die Möglichkeit, einen Termin im Attribut einer Entität mit einem Faktor zu multiplizieren und einen anderen Wert hinzu zu addieren. Über die Methoden des Interfaces **NeIValueReference** können Sie z.B. die Werte als Double- oder Long-Werte aus den Attributen holen oder feststellen, ob der Wert von einem anderen Attributwert abhängig ist.

Wir stellen Ihnen in dem Programmierbeispiel unten eine freie Layergestaltung vor, bei der Sie beispielhaft die Verwendung von ValueReference-Objekten mitverfolgen können. Als Ergebnis werden drei Layer erstellt: einen linienförmigen für geplante Vorgänge und zwei Rechtecklayer, die für einen laufenden Vorgang mit Fertigstellungsgrad zu einem Knoten zusammengesetzt werden. Zudem werden die Layer mit dekorativen Elementen wie Symbolen und Beschriftungen versehen:



Das Code-Beispiel:

```
// Notwendige Klassen:
import de.netronic.common.beanbase.*;
...

// Das Knoten-Design für normale Knoten wird abgeschaltet
jGantt1.setNodeDesign(null);
```

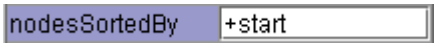
```
// Area Style definieren
NeAreaStyle transparentGray = new NeAreaStyle (new Color (100, 100, 100,
100), 20, Color.gray);

// ValueReference - Objekte anlegen

// Die ValueReferenz startRef ist eine normale Referenz zum
// Datumsattribut start und enthält das Anfangsdatum
startRef = new NeValueReference ("start");

// Die ValueReference endRef enthält das Enddatum und ist definiert als
// Summe aus startRef + dem Inhalt des Attributs duration
endRef = new NeSumValueReference ("duration",
NeSumValueReference.DURATION_FACTOR_DAY, start);

// pcRef ist definiert als Prozentwert zwischen startRef und endRef
// das Attribut pc enthält den Prozentwert
pcRef = new NeRelativeValueReference ("pc");
pcRef.setBaseValueReference(startRef);
pcRef.setReferenceValueReference(endRef);
// Umrechnung Prozentwert in Faktor
pcRef.setFactor(0.01);


// Dekorationsobjekte anlegen: Beschriftungen + Label-Objekte

// Beschriftung unterhalb des Vorgangs
NeAnnotation bottomAnn = new NeAnnotation();
bottomAnn.setFormatString("Vorgang: {0}\n   dauert:  {1} Tage!");
bottomAnn.setDataFieldIDs(new String []{"description", "duration"});
bottomAnn.setBorderStyle(Color.white);
bottomAnn.setBackgroundStyle(transparentGray);
bottomAnn.setScaleFontToFit(true);

//Blaues Dreieck
NeSymbol triangleLeft = new NeSymbol (6);
triangleLeft.setLineStyle(Color.blue);
triangleLeft.setAreaStyle(null);
triangleLeft.setExtension(new Dimension (400, 400));

//Rotes Dreieck
NeSymbol triangleRight = new NeSymbol (5);
triangleRight.setLineStyle(Color.red);
triangleRight.setAreaStyle(null);
triangleRight.setExtension(new Dimension (400, 400));


// Filter anlegen: Die beiden Filter, die letztlich zur Anwendung kom-
men, // werden aus drei Basisfiltern kombiniert

// Basisfilter: für alle Entitäten des Entitäten-Sets für Knoten
// Filter ist vorgefertigt
NeIFilter forNodes = new JGEntitySetFilter (jGantt1.getNodeSetName());

// Basisfilter: nur für begonnene Vorgänge
// Filter wird selbst erstellt
NeIFilter started = new NeFilterBase()
{
  public boolean apply (NeIEntity entity)
```

```
{
  double percentage = entity.getValueAsDouble("pc");
  return (percentage>0.0 && percentage < 100.0);
}
};

// Basisfilter nur für nicht begonnene Vorgänge
// Filter wird selbst erstellt
NeIFilter planned = new NeFilterBase()
{
public boolean apply (NeIEntity entity)
  {
  return entity.getValueAsDouble("pc") <= 0.0;
  }
};

// Kombinierter Filter für nicht begonnene Vorgänge des
// Knoten-Entitäten-Sets
NeIFilter plannedNodes = new NeCombinedFilter(planned, "&&", forNodes);

// Kombinierter Filter für begonnene Vorgänge des Knoten-Entitäten-Sets
NeIFilter startedNodes = new NeCombinedFilter(started, "&&", forNodes);


// Layer-Definitionen anlegen

// geplante Vorgänge
JGIGanttGraph ganttGraph=jGantt1.getGanttGraph();
NeILayerDefinition plannedLayer = ganttGraph.createLayerDefinition

("planned");
plannedLayer.setStartValueReference(startRef);
plannedLayer.setEndValueReference(endRef);
plannedLayer.setLayerType(NeILayerDefinition.LAYERTYPE_LINE);
plannedLayer.setFilter(plannedNodes);
plannedLayer.attachLabel (triangleRight,
NeILabelAttachment.POINT_CENTER_LEFT, null);
plannedLayer.attachLabel (triangleLeft,
NeILabelAttachment.POINT_CENTER_RIGHT, null);

// begonnene Vorgänge: durchsichtiger Balkenlayer
NeILayerDefinition startEnd = gantt-
Graph.createLayerDefinition("startEnd");
startEnd.setStartValueReference(startRef);
startEnd.setEndValueReference(endRef);
startEnd.setFilter(startedNodes);
startEnd.setHeight(300);
startEnd.setBottomPicture(bottomAnn);
startEnd.setTopPicture(triangleLeft);

// orange-schwarzer Balkenlayer, der den Ausführungsgrad anzeigt
NeILayerDefinition pc = ganttGraph.createLayerDefinition("pc");
pc.setFilter(startedNodes);
pc.setStartValueReference(startRef);
pc.setEndValueReference(pcRef);
pc.setHeight(150);
pc.setAreaStyle(new NeAreaStyle (Color.orange, 1, Color.black));
pc.setDrawingPriority(100);
```

Sie haben nun die freie Gestaltung von Layern programmiert und sie zu Knoten zusammengesetzt. Wir werden uns nun dem Verhalten von Knoten im Gantt-Graphen zuwenden.

5.3.3 Die Sortierung von Knoten

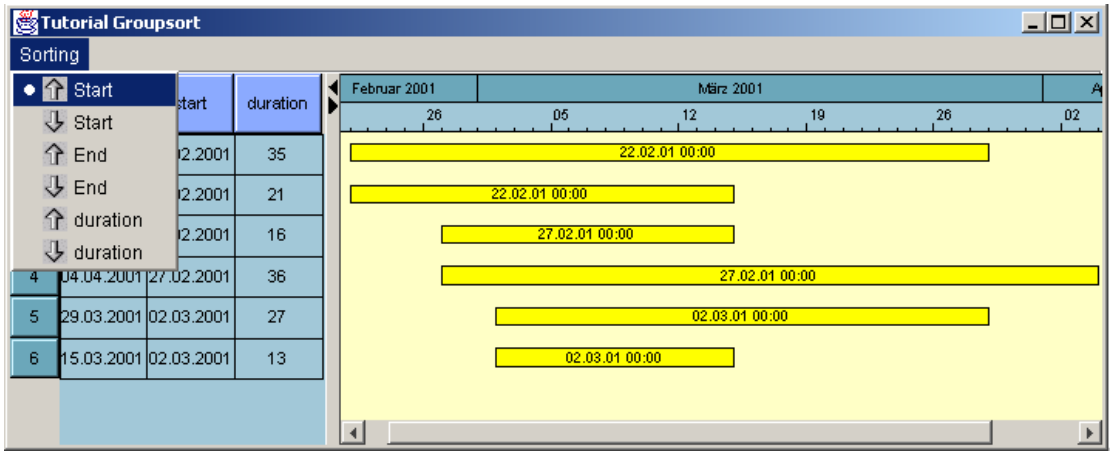
Häufig kommt es in Gantt-Diagrammen vor, dass Knoten nicht in der Reihenfolge angezeigt werden sollen, in der sie anfänglich erstellt wurden. Werden Vorgänge z.B. vom Benutzer in aufsteigender Reihenfolge ihres Anfangsdatums eingegeben, kann später eine Ansicht gewünscht sein, in der Knoten z.B. in aufsteigender Reihenfolge des Enddatums angeordnet sind. Ein Gantt-Diagramm sollte daher Vorgänge ordnen und umordnen können. In diesem Kapitel zeigen wir, wie Sie mit VARCHART JGantt Knoten nach verschiedenen Kriterien sortieren. Zu diesem Thema steht Ihnen das mitgelieferte Beispielprogramm **groupsort.java** zur Verfügung. Die Sortierung von Knoten mit VARCHART JGantt erfolgt über einen einzigen Aufruf. Mit ihm geben Sie dem JGantt-Objekt das Attribut der Knoten-Entität bekannt, nach dessen Inhalt sortiert werden soll. Weiterhin bestimmen Sie mit diesem Aufruf, ob in aufsteigender oder absteigender Reihe sortiert werden soll. Sie können die Eigenschaft im Design-Modus setzen:



... oder im Quell-Code:

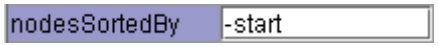
```
//Sortierung aufsteigend nach Anfangsdatum, d.h. Attribut "start"
 jGantt1.setNodesSortedBy(new NeEntityComparator ("start",false));
```

Im Design-Modus legen Sie über das Vorzeichen die Sortier-Reihenfolge fest. Bei „+“ stehen die großen Werte oben, die kleinen unten, bei „-“ umgekehrt. Hinter dem Vorzeichen geben Sie den Namen des Attributs an, nach dessen Inhalt sortiert werden soll. Im Quellcode wird über die Methode **setNodesSortedBy** eine Instanz der Klasse **NeEntityComparator** angelegt, die die Inhalte des Attributs „start“ vergleicht. Der zweite Parameter zeigt die Reihenfolge an. Bei „false“ stehen die großen Werte oben, die kleinen unten, bei „true“ wird umgekehrt sortiert. Im Diagramm von **sorting.java** sieht das Ergebnis mit den Parametern **start** und **false** folgendermaßen aus:

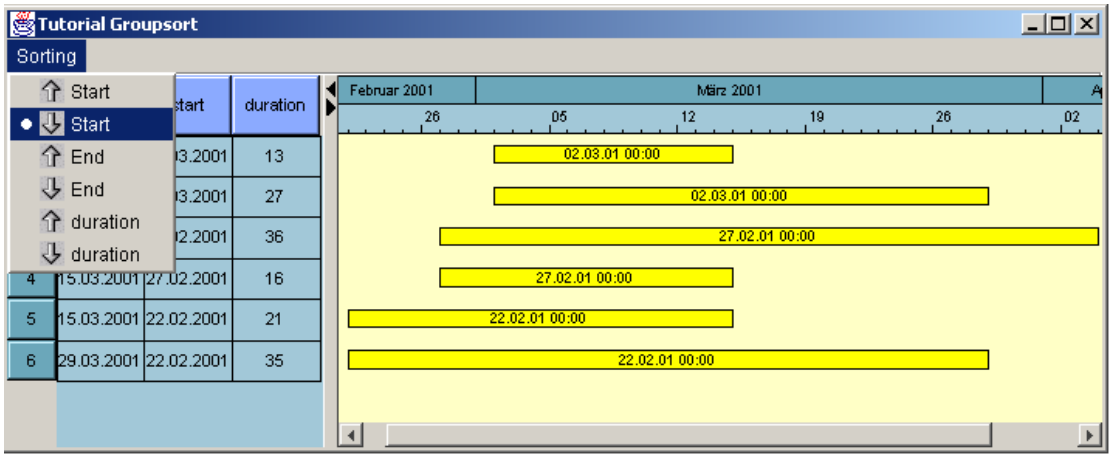


Das Beispielprogramm erlaubt über ein kleines Menü, zwischen den Feldern **Start**, **Ende** und **Duration** in jeweils auf- und absteigender Reihenfolge zu wählen. Gleichzeitig werden die Anfangsdaten im Knoten abgebildet, so dass Sie die Reihenfolge erkennen können. Sie ist hier aufsteigend: oben (d.h. im ersten Knoten) stehen die kleinen Werte des Anfangsdatums, unten (im letzten Knoten) die großen.

Hier das Beispiel mit den Parametern **start** und **true**:




```
//Sortierung absteigend nach Anfangsdatum, d.h. Attribut "start"
jGantt1.setNodesSortedBy(new NeEntityComparator ("start",true));
```



Die Sortierung erfolgt absteigend nach dem Anfangsdatum: oben (im ersten Knoten) stehen die großen Werte, unten (im letzten Knoten) die kleinen.

Damit kennen Sie nun die Grundlagen der Sortierung von Knoten. Eine komplexere Form der "Sortierung" ist das Gruppieren von Knoten nach bestimmten Kriterien, das Sie im folgenden Kapitel kennenlernen.

5.3.4 Knoten – Gruppierung oder Hierarchie

Mit VARCHART JGantt ist es möglich, die Darstellung der Knoten zu strukturieren. Dabei gibt es zwei verschiedene Möglichkeiten, die grundlegende Unterschiede, aber auch einige Gemeinsamkeiten aufweisen:

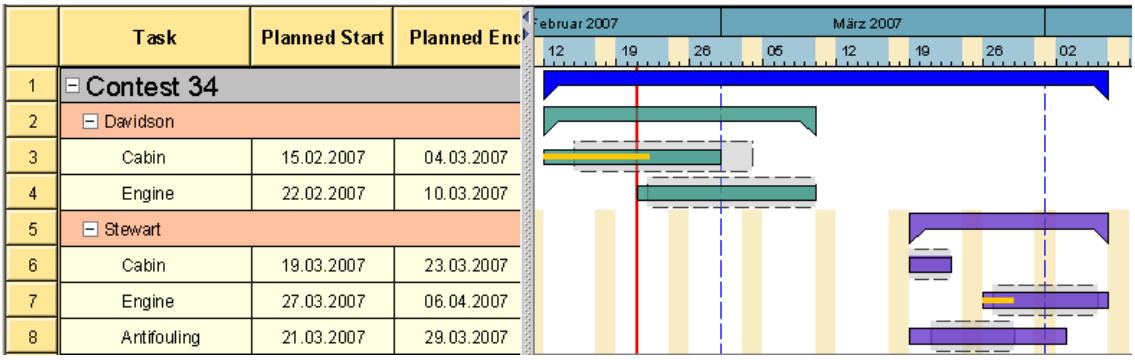
- das Gruppieren von Knoten
- das hierarchische Anordnen von Knoten

Wir werden zunächst die Gemeinsamkeiten und Unterschiede und die damit verbundenen Konsequenzen für Ihre Anwendung beschreiben, bevor wir zur eigentlichen Programmierung übergehen.

Die Gruppierung von Knoten ermöglicht es, Vorgänge, die ein gemeinsames Merkmal haben, zusammenzufassen: Bei einer Raumbelegung kann es sich um Räume derselben Etage, bei einer Fuhrparkverwaltung um Kraftwagen desselben Standortes, bei einer Personaleinsatzplanung um Mitarbeiter desselben Teilprojektes handeln. Das Gruppieren von Knoten ist eine häufig gestellte Anforderung an Gantt-Diagramme.

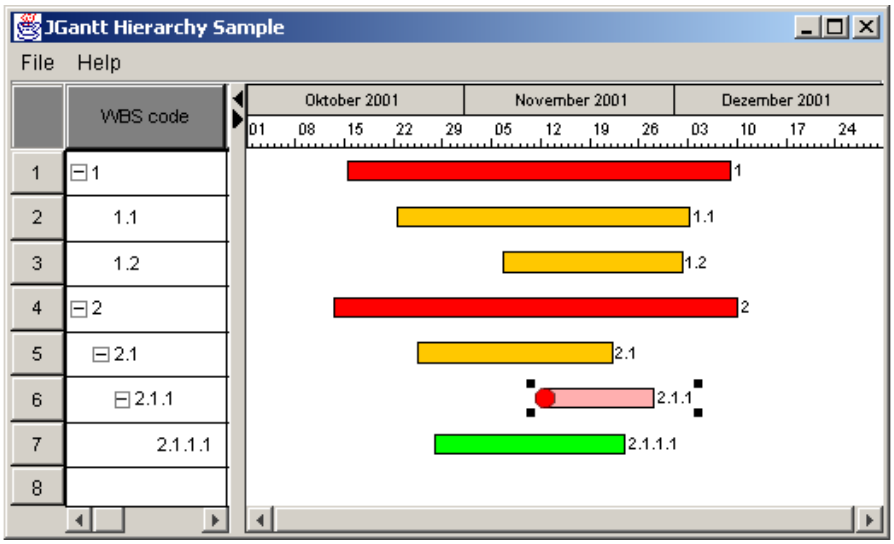
Eine äußerliche Gemeinsamkeit der Gruppen mit den Ebenen einer Hierarchie ist es, dass sie einander über- oder untergeordnet sein können. So können zum Beispiel in einem Projekt Arbeitsvorgänge nach Bearbeitern und nach Teilprojekten gruppiert werden. Diese beiden Kriterien können voneinander abhängig gemacht und damit die Gruppierungsebenen übereinander angeordnet werden; in diesem Fall würden die Arbeitsvorgänge für jeden Bearbeiter eines Teilprojektes angezeigt. Näher betrachtet, ist der wesentliche Unterschied zu einer Hierarchie jedoch, dass jede Gruppe, unabhängig davon, ob sie auf gleicher, höherer oder niedrigerer Ebene angesiedelt ist, nach einem eigenen Kriterium gebildet werden kann. Im genannten Beispiel wurden die Ebenen nach den Kriterien **Bearbeiter** und **Teilprojekt** gebildet; sie sind verschieden und zeichnen eine Gruppierung aus.

Im Bild unten heißt das Teilprojekt beispielhaft „Contest 34“ und die Bearbeiter „Davidson“ und „Stewart“:



Mehrfachgruppierung nach Teilprojekt und Bearbeiter

In einer Hierarchie hingegen werden alle Ebenen nach einem vordefinierten Hierarchiecode, also ein- und demselben Kriterium angelegt. Sie können bei der Hierarchie das Kriterium, nach dem Knoten auf eine bestimmte Ebene gesetzt werden, nicht frei definieren. In der Praxis wenden Sie die Hierarchie z.B. für die Abbildung einer komplexen Montage an, die in einen festen Fertigungsprozess eingegliedert ist.



Gliederung nach Hierarchiecode: oberste Ebene einstellig, nächste Ebene zweistellig, etc...

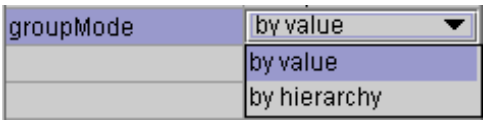
Bei Gruppen können Sie die Knoten sowohl innerhalb der Gruppe als auch die Gruppen selbst sortieren. In der Hierarchie können Sie die Knoten nicht sortieren, da die Reihenfolge durch den Hierarchiecode festgelegt ist. Der Anwender kann die Knoten verschieben; entsprechend ändert sich der Hierarchiecode des verschobenen Knotens, da der Hierarchie-Code immer in derselben Ordnung bleibt. Sie können auch bei der Gruppierung Ihre Anwendung so programmieren, dass untergeordnete Knoten von einer in die andere

Gruppe verschiebbar (und dabei ihr Aussehen ändern) sind. Sie können so z.B. Aufträge bestimmten Maschinen zuweisen oder Tätigkeiten an Personen koppeln. Sie können bei der Gruppierung auf der untersten Ebene mehrere Knoten pro Zeile darstellen. Das Layout können Sie so optimieren, dass die Knoten einer Gruppe sich bei der Darstellung nicht überlappen. Sie können bei der Hierarchie nicht mehrere Knoten pro Zeile darstellen; somit erübrigt sich die Layoutoptimierung.

Wir stellen Ihnen im nächsten Kapitel die Gruppierung von Knoten vor und die Einbindung in die Hierarchie im übernächsten. Für das folgende Kapitel finden Sie in der mitgelieferten Beispielsammlung das Programm **grouping.java**, das die einfache Gruppierung von Knoten exemplarisch vorführt.

5.3.5 Gruppierung

Mit der Eigenschaft **groupMode** legen Sie zunächst fest, dass Sie die Gruppierung durchführen wollen und diese über ein Knotenattribut erfolgen soll:



..und im Quellcode:

```
//Gruppierung über Knotenattribut
jGantt1.setGroupMode(de.netronic.jgantt.JGantt.GROUP_BY_VALUE);
```

Nun legen Sie das Attribut an, benennen und füllen es (wie dies erfolgt, wurde im Kapitel über das Anlegen von Datenstrukturen und Daten bereits beschrieben). Es soll in unserem Beispiel **group** heißen. Alle Entitäten der Knoten eines Sets, die in diesem Attribut denselben Inhalt zeigen, gehören zur selben Gruppe. Verschiedene Inhalte in diesem Feld führen zur Bildung unterschiedlicher Gruppen.

Bei der Bildung einer Gruppe kann es ausreichen, die Mitglieder einer Gruppe auf die beschriebene Art zu kennzeichnen. Sie sind damit alle gleichwertig. Es kann aber auch sinnvoll sein, zum einen die untergeordneten Knoten einer Gruppe einzurichten, zum anderen einen übergeordneten Knoten mit Meta-Informationen der Gruppe hinzuzufügen.

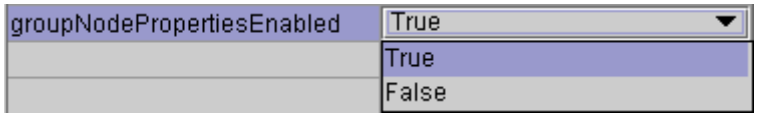
Für diesen Fall definieren Sie ein besonderes Entitäten-Set, dessen Entitäten wir als **Gruppen-Entitäten** bezeichnen, aus dem die grafischen Entsprechungen generiert werden, die **Gruppenknoten**. Diesem Set weisen Sie einen Namen zu; für unser Beispiel wählen wir den Namen **Groups**. Das Gruppen-Set kann eine andere Datenstruktur aufweisen als das der untergeordneten Knoten, muss aber dasselbe Gruppierungsattribut besitzen, das ja in unserem Fall **group** heißt. Dieses Set definieren Sie als Entitäten-Set für Gruppenknoten, indem Sie seinen Namen dem JGantt-Objekt bekannt geben:

```
//Gruppen-Entität bekanntgeben
jGantt1.setGroupNodeSetName("Groups");
```

Mit dieser Bekanntgabe steht dem Entitätenset „Groups“ nun die gesamte Palette der Gruppenknoten-Eigenschaften zur Verfügung:

groupBy	
groupHierarchyBy	
groupLayout	multiple rows
groupMode	by value
groupNodeAnnotations	
groupNodeColorScheme	blues
groupNodeDateFormat	
groupNodeDates	
groupNodeDesign	Summary
groupNodeFont	"Dialog", 1, 7
groupNodeProfile	
groupNodeProfileBy	
groupNodePropertiesEnabled	True
groupNodeSetName	
groupNodeSymbols	--- none ---, --- none ---
groupNodeZeroLengthSymbol	--- none ---
groupNodeZeroLengthVisible	False

Sie können diese Eigenschaften allerdings auch in ihrer Gesamtheit abschalten:




```
// Gruppenknoten-Eigenschaften anschalten
jGantt1.setGroupNodePropertiesEnabled(true);
```

Mit den Eigenschaften können Sie für die Gruppenknoten nun ähnlich viele Optionen nutzen wie für die untergeordneten Knoten. Einige abweichende Details werden im Verlauf dieses Kapitels noch besprochen.Die Zugehörigkeit zu einer Gruppe sowohl der untergeordneten Entitäten als auch der Gruppen-Entität haben Sie über das gemeinsame Attribut „group“ und seinen Inhalt festgelegt. Setzen Sie bitte als Inhalt für verschiedene Gruppen z.B. die Buchstaben „A“, „B“, „C“ etc. Für jede Gruppe richten Sie jeweils mehrere untergeordnete Knoten und einen Gruppenknoten ein. Anschließend teilen Sie dem JGantt-Objekt mit, dass die Gruppenbildung über das Attribut **group** erfolgen soll. Dies können Sie im Design-Mode...

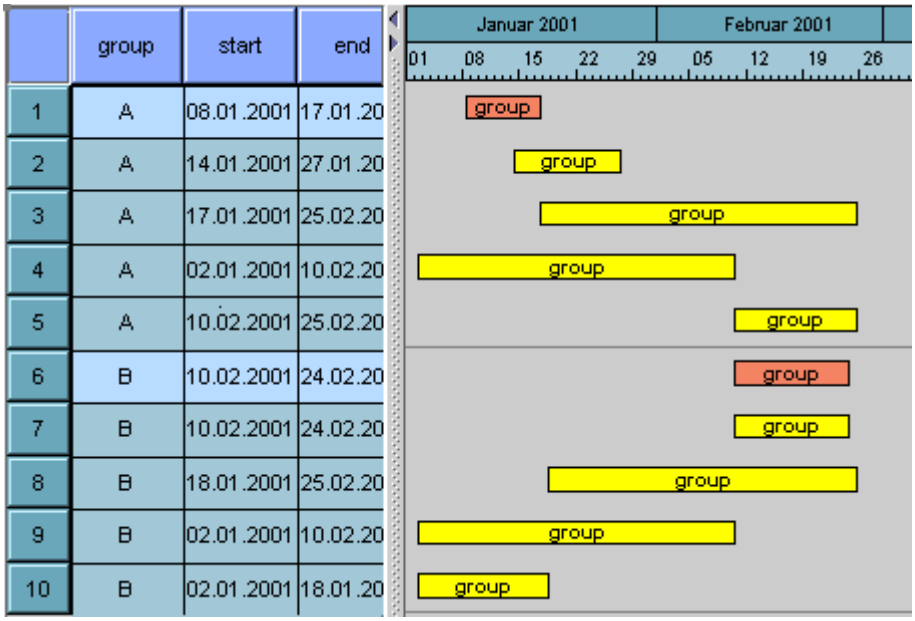
... oder entsprechend im Quell-Code vornehmen:

```
//Gruppierungsattribut bekanntgeben
jGantt1.setGroupBy("group");
```

Wenn Sie in der Tabelle das Attribut **group** in einer Spalte darstellen,

```
//Tabellenspalten setzen
jGantt1.setTableColumns(new String[]{"group", "start", "end"});
```

erscheint etwa folgendes Bild im Design-Mode:



Die Tabelle zeigt in der Spalte ganz links den Inhalt des Attributs **group** an, während die Knoten im Gantt-Graphen seinen Namen anzeigen. Zur Laufzeit zeigen die Knoten ebenfalls den Inhalt von **group**. Erkennbar sind im Gantt-Graphen zwei Gruppen: eine obere und eine untere, getrennt durch eine waagerechte Linie. Jede Gruppe ist durch einen dunklen Gruppenknoten und mehrere helle untergeordnete Knoten gekennzeichnet. In der Tabelle sind die Gruppen an den Inhalten des Attributs **group** erkennbar: „A“ und „B“. Weiterhin zeigt auch hier der Gruppenknoten eine andere Farbe als die untergeordneten Knoten. In der Tabelle zeigen Gruppenknoten die **shadedColor**, während untergeordnete Knoten in der **shadedAlternateColor** des Tabellen-Farbschemas eingefärbt sind.

```
//Tabellen-Farbschema setzen
jGantt1.setTableColorScheme
              (de.netronic.jgantt.JGColorScheme.getInstance("blues"));
```

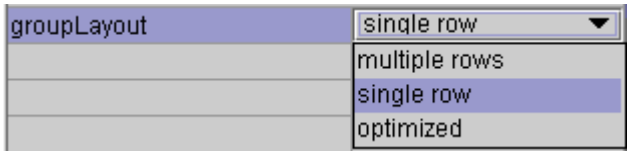
Im Gantt-Graphen ist die Farbe der untergeordneten Knoten durch das Knoten-Farbschema und die Farbe der Gruppenknoten durch das Gruppenknoten-Farbschema festgelegt:

```
//Knoten-Farbschema setzen
jGantt1.setNodeColorScheme
           (de.netronic.jgantt.JGColorScheme.getInstance("contrast"));

//Gruppenknoten-Farbschema setzen
jGantt1.setGroupNodeColorScheme
          (de.netronic.jgantt.JGColorScheme.getInstance("red apple"));
```

Sie können auch jedem Gruppenknoten eine eigene Farbe zuweisen. Die Details dazu erfahren Sie im Kapitel über dynamische Farbgebung. Falls Sie die Darstellung wie im obigen Bild nicht erhalten haben, könnte dies am Gruppen-Layout liegen. Sie haben die Möglichkeit, Gruppen auf drei verschiedene Weisen anzuordnen:

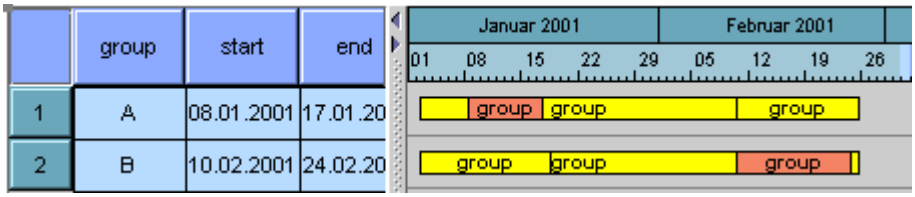
1. jeden untergeordneten und jeden Gruppenknoten in einer eigenen Zeile (multiple rows)
2. alle Knoten einer Gruppe in einer Zeile, wobei sich die Knoten überlagern können (single row)
3. alle Knoten einer Gruppe in einer Zeile, ohne Überlagerung (optimized)

Die erste Option ist in dem obigen Beispiel abgebildet. Sie können das Gruppenlayout folgendermaßen ändern:

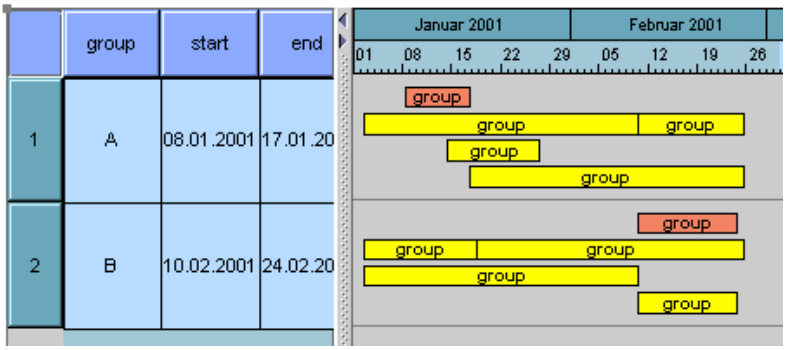



```
//Gruppenlayout festlegen
jGantt1.setGroupLayout(JGantt.GROUP_ROWLAYOUT_SINGLE);
```

Das Ergebnis für die Option **single row** ist für unser Beispiel nicht sinnvoll und zeigt unvorteilhafte Überlagerungen:

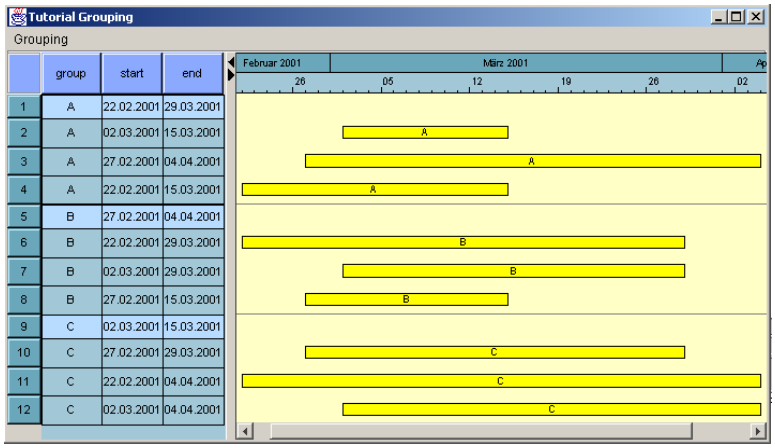


Beim Ergebnis für die Option **optimized** (Bild unten) befinden sich ebenfalls alle Knoten in einer Zeile, allerdings wird die Überlagerung vermieden. Der Unterschied zur Option **multiple rows** liegt darin, dass bei **multiple rows** jeder Knoten einer Tabellenzeile gegenübersteht und somit alle untergeordneten und Gruppenknoten auch in der Tabelle dargestellt werden, während bei **optimized** alle Knoten der Gruppe einer einzigen Tabellenzeile gegenüberstehen und in der Tabelle nur der Gruppenknoten dargestellt wird:

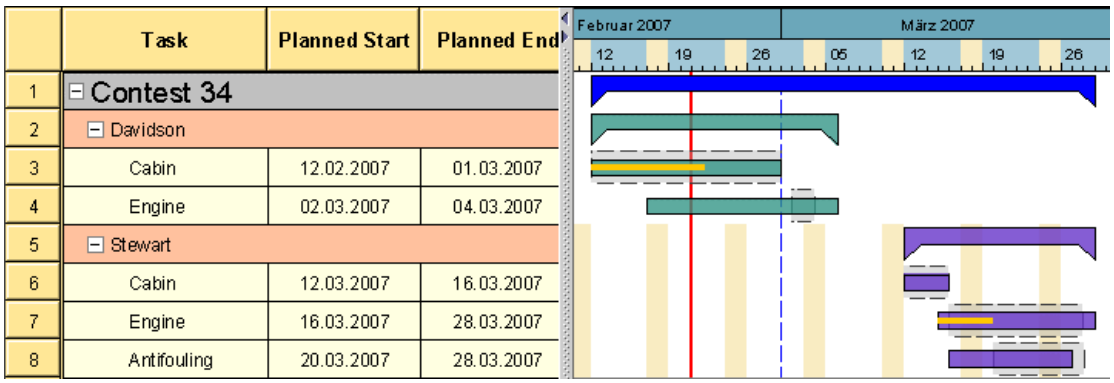


Sie können übrigens die Darstellung der Gruppenknoten im Gantt-Graphen unterdrücken, indem Sie die Bekanntgabe von Anfangs- und Endtermin des Gruppenknotens revidieren. Dazu klicken Sie bei der Eigenschaft **groupNodeDates** in das Feld mit den drei Punkten. Sie gelangen zum Dialog **GroupNodeDates**, in dem Sie die gesetzten Termine entfernen (bei Einzelmarkierung) oder löschen (alle, ohne Markierung) können:

Das Ergebnis zur Laufzeit sieht folgendermaßen aus:



In der Tabelle werden die Daten des Gruppenknotens dargestellt, im Gantt-Graphen wird seine Darstellung unterdrückt. Sie können Knoten auch auf mehreren Ebenen gruppieren:



In dem Beispiel oben wurden die Arbeitsvorgänge nach Teilprojekten und Mitarbeitern gruppiert. In der Tabelle ist erkennbar, dass das Teilprojekt "Contest 34" und die dazugehörigen Mitarbeiter "Davidson" und "Stewart" heißen. Im Diagramm rechts wurde die Farbgebung entsprechend über dynamische Farbgebung programmiert. Die Mehrfachgruppierung erfordert für jede Gruppierungsebene ein weiteres Attribut in der Gruppenknotenentität. Für das obige Beispiel wurden diese "Teilprojekt" und "Mitarbeiter" genannt. "Mitarbeiter" führt die Knoten der unteren Ebene zusammen, "Teilprojekt" die Knoten auf der oberen Ebene. Alle Basis- und Gruppenknoten-Entitäten besitzen diese beiden Attribute. Der Code für die Mehrfachgruppierung sieht so aus:

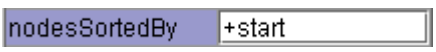
```
//Zwei Gruppierungsattribute setzen
jGantt1.setGroupBy("Teilprojekt, Mitarbeiter");
```

Die erste Gruppierungsebene wird folgendermaßen gebildet: Die Gruppenknoten-Entität des Mitarbeiters "Stewart" und die zugehörigen Basisknoten-Entitäten besitzen den Wert "Stewart" im Attribut "Mitarbeiter"; die Gruppenknoten-Entität des Mitarbeiters "Davidson" sowie die zugehörigen Basisknoten-Entitäten besitzen den Wert "Davidson". Das Attribut "Mitarbeiter" in der Gruppenknoten-Entität "Contest 34" ist leer. Daran ist erkennbar, dass dieser Gruppenknoten nicht zur ersten Ebene gehört.

Die zweite Gruppierungsebene wird folgendermaßen gebildet: Alle Basisknoten und alle Gruppenknoten der ersten und zweiten Ebene besitzen im Attribut "Teilprojekt" den Wert "Contest 34".

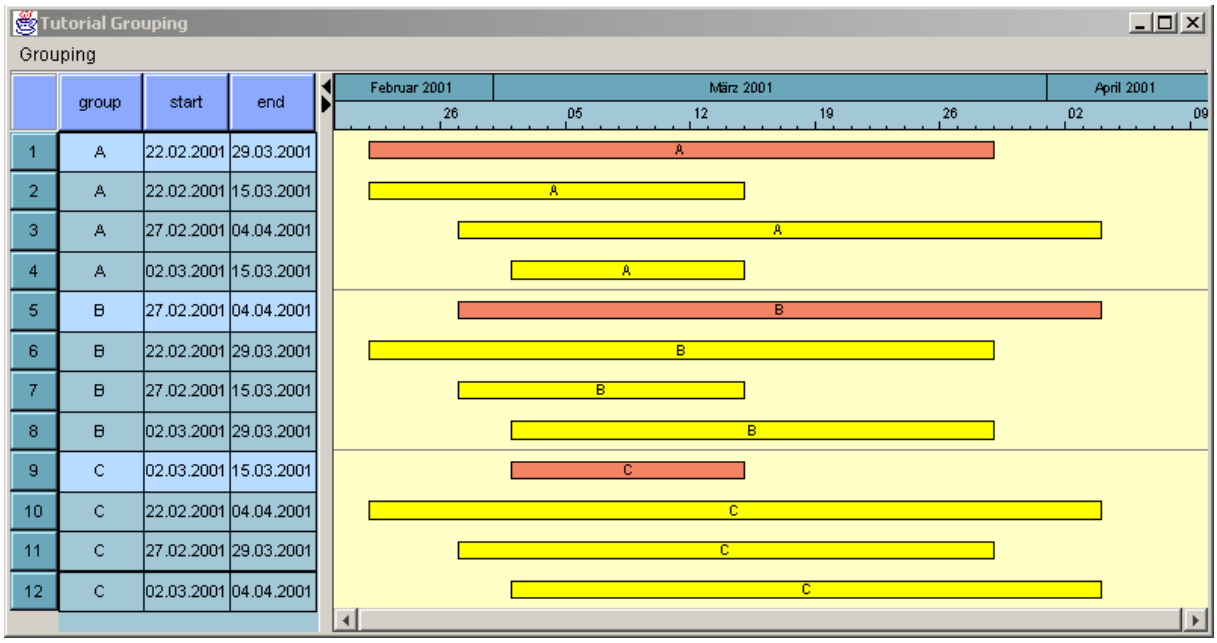
Damit haben Sie nun die grundlegenden Kenntnisse für die Programmierung der Mehrfachgruppierung erworben und wir können nun die Sortierung von Gruppen behandeln.

Gruppierte Knoten zu sortieren ist eine häufige Anforderung an Gantt-Diagramme. Sie können in VARCHART JGantt bei Gruppenbildung die Knoten innerhalb der Gruppe nach einem Kriterium sortieren, oder die Gruppen selbst. Beim Sortieren der Knoten innerhalb der Gruppe wird der Gruppenknoten von der Sortierung ausgenommen und steht immer zuoberst. Im untenstehenden Beispiel wurde aufsteigend nach dem Anfangsdatum sortiert, wodurch die kleinsten Anfangsdaten oben in einer Gruppe (direkt unterhalb des Gruppenknotens), die größten unten in der Gruppe stehen:

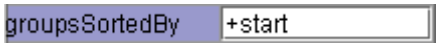


... oder im Quell-Code:

```
//Sortierung aufsteigend nach Anfangsdatum, d.h. Attribut "start"
jGantt1.setNodesSortedBy(new NeEntityComparator ("start",false));
```



Die Sortierung der Gruppen erfolgt über die Eigenschaft:



... und entsprechend im Quellcode:

```
//Sortierung der Gruppen aufsteigend nach Anfangsdatum= Attribut "start"
jGantt1.setGroupsSortedBy(new NeGroupComparator ("start",false));
```

Die Gruppenknoten werden aufsteigend nach dem Inhalt des Attributs **start**, d.h. nach dem Anfangsdatum ihrer Gruppenknoten-Entität sortiert, wodurch die frühesten Anfangsdaten oben, die spätesten unten stehen.

Wir haben Ihnen in diesem Kapitel das Gruppieren und die Sortierung bei Gruppen vorgestellt. Wir zeigen Ihnen jetzt noch, wie Sie bei der Darstellung einer Gruppe die Länge des Gruppenknotens automatisch an die Gesamtlänge der Gruppe anpassen können, so dass die Zeitspanne des Gruppenknotens alle Zeitspannen der Gruppenmitglieder umfasst. Es ist sogar möglich, Gruppenknoten, die fehlen, zu identifizieren und zu ergänzen. Dazu verwenden Sie das Interface des NeIGroupValueUpdaters:

```
// Holen des GroupValueUpdater-Objektes
myUpdater = jGantt1.getGroupValueUpdater();
```

```
// Bezeichnung der Ziel- und QuellAttribute für Anfangs- und Enddatum:
//Generierung des Gruppenknotenstartdatums aus den Startdaten der zuge-
hörigen Basisknoten
myUpdater.addReference (startDateOfGroupNode, startDateOfNode, myUpdat-
er.MINIMUM)
//Generierung des Gruppenknotenenddatums aus den Enddaten der zughörigen
Basisknoten
myUpdater.addReference (endDateOfGroupNode, endDateOfNode, myUpdat-
er.MAXIMUM)

//Anlegen von fehlenden Gruppen einschalten:
setCreateMissingGroupNodes(true);

// Aktivierung
myUpdater.setActive(true);
```

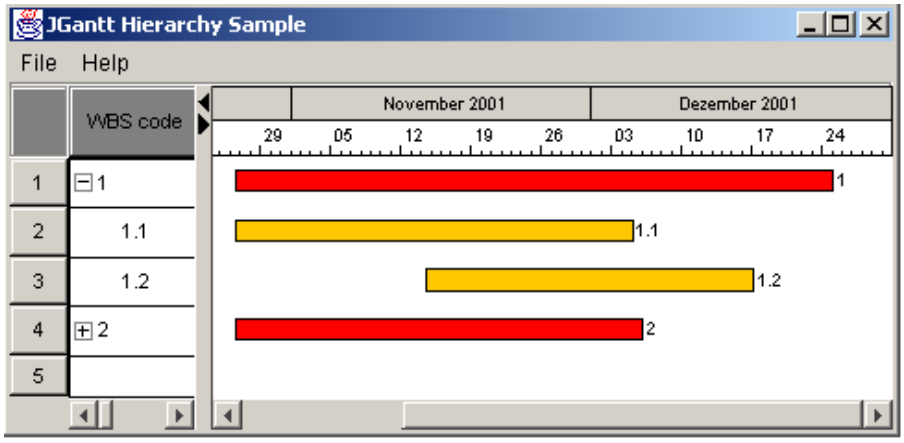
Es werden nun die Startdaten aller Basisknoten, die zur Gruppe gehören, in die Berechnung des Startdatums des Gruppenknotens miteinbezogen, Entsprechendes gilt für das Enddatum. Jede Datumsveränderung eines Basisknotens wird auch im Datum des Gruppenknotens nachgezogen (natürlich auch in Gruppenknoten höherer Ebenen), es sei denn, Sie deaktivieren den GroupNodeUpdater wieder:

```
// Deaktivierung
myUpdater.setActive(false);
```

Damit haben Sie die wesentlichen Dinge bei der Programmierung von Gruppen kennengelernt.

5.3.6 Hierarchische Strukturen bei Knoten

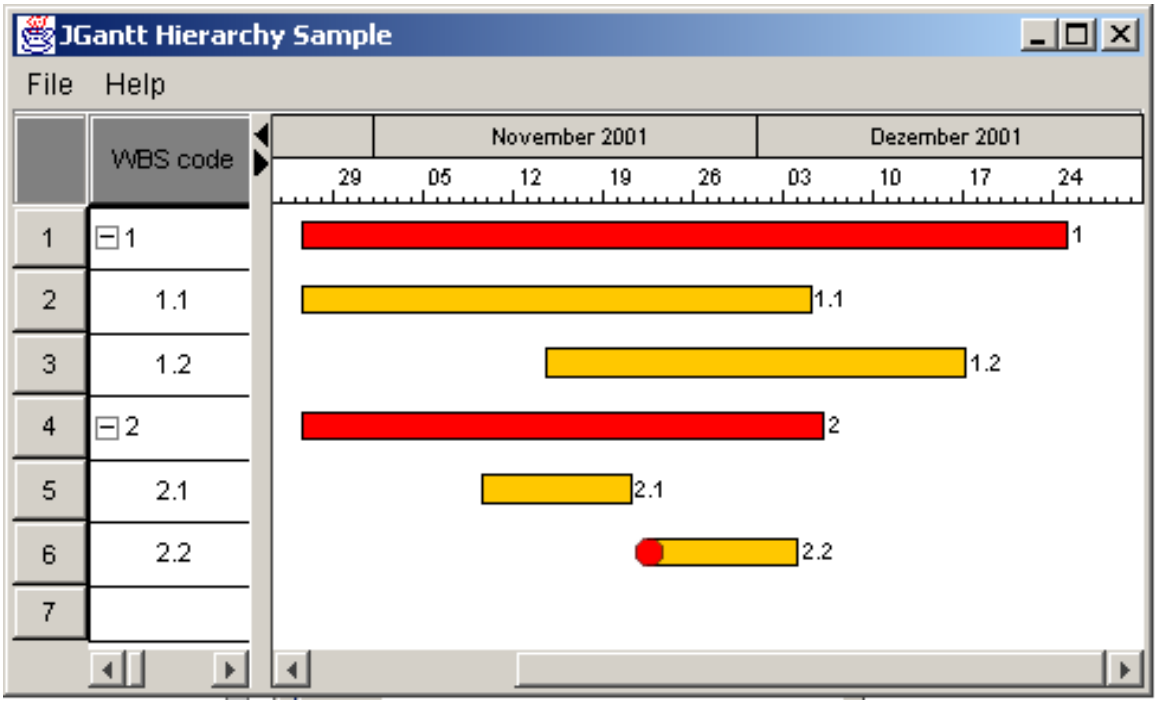
Bei komplex strukturierten Abläufen kann es notwendig werden, die Vorgänge in hierarchischen Strukturen abzubilden. Diese sind im Unterschied zu mehrstufiger Gruppierung durch den Hierarchie-Code strukturiert. Eine hierarchische Darstellung der Vorgänge in der Tabelle ermöglicht das Kollabieren und Expandieren von Teilbereichen des Diagramms und gestattet in der Tabelle eine der Hierarchie entsprechende Einrückung des Textes in einer wählbaren Spalte. Das Kollabieren und Expandieren von Teilbereichen des Diagramms trägt zur Verbesserung der Übersichtlichkeit in Gantt-Diagrammen mit bei. Zum Thema „Hierarchie“ ist ein Beispielprogramm mit dem Namen **hierarchy.java** im Lieferumfang enthalten, das den in diesem Kapitel verwendeten Quell-Code enthält.



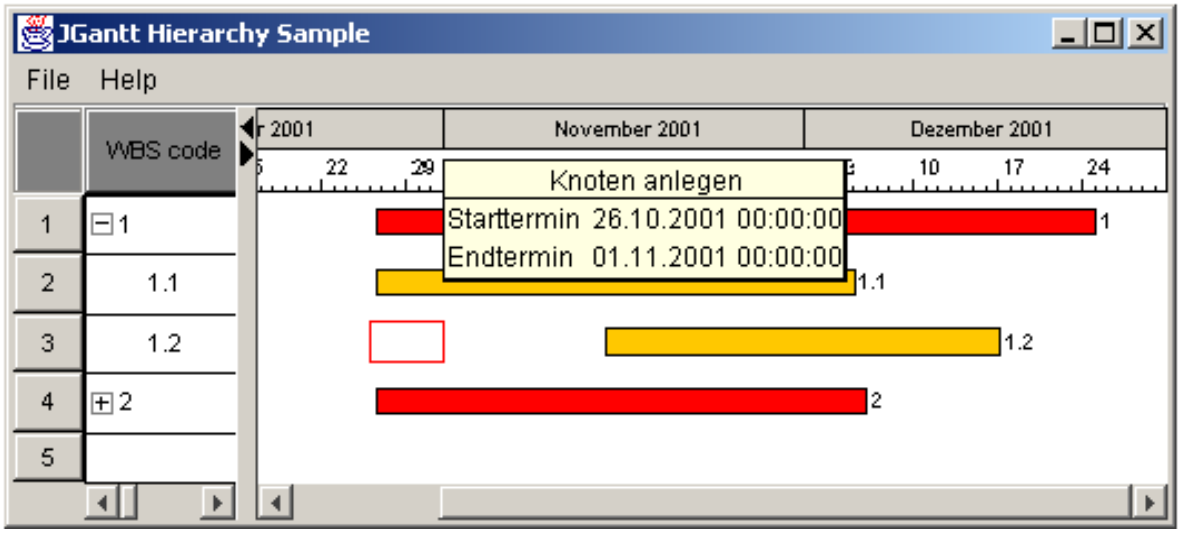
Im Bild oben zeigen die Ziffern in der Tabelle durch Einrückung die Hierarchie-Ebene an. Vor Knoten 1 der „obersten“, d.h. der am weitesten nach links gerückten Ebene steht ein Minuszeichen, das anzeigt, dass die abhängigen Knoten expandiert, d.h. in der Tabelle und im Gantt-Graphen sichtbar sind. Unterhalb der expandierten Knoten folgt Knoten 2 der „obersten“ Ebene. Vor Ziffer 2 steht ein Pluszeichen, das anzeigt, dass die von Knoten 2 abhängigen Knoten kollabiert und somit unsichtbar sind. Die abgebildeten Ziffern stellen den sogenannten Hierarchie-Code dar. Er beschreibt die Abhängigkeiten der Knoten untereinander. Er wird nicht zwangsläufig in einem Gantt-Diagramm angezeigt, wurde jedoch im obigen Beispiel zu Demonstrationszwecken sichtbar gemacht.

Die Hierarchie aus der Sicht des Anwenders

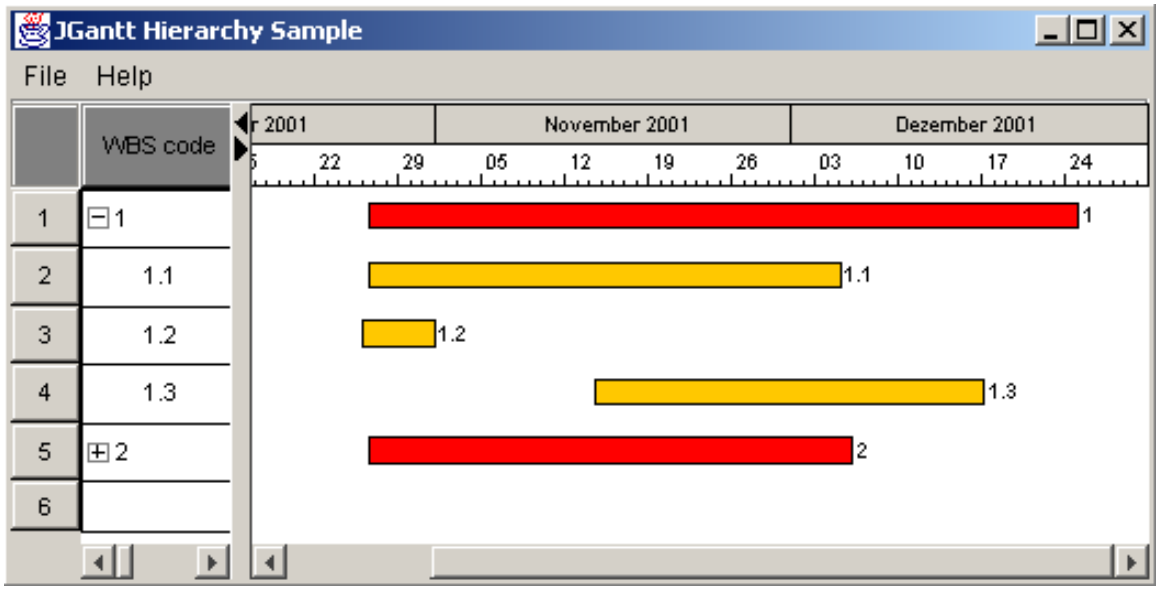
Bevor Sie mit der Programmierung beginnen, stellen wir Ihnen die wichtigsten Funktionalitäten und möglichen Interaktionen der Hierarchie vor. Nur Knoten mit untergeordneten Knoten zeigen ein Plus- oder Minuszeichen. Durch Anklicken können diese ineinander umgewandelt und die zugehörigen Teilhierarchien entsprechend expandiert oder kollabiert werden. Im Beispiel unten wurde Knoten 2 der obersten Ebene expandiert und zeigt nun zwei weitere Knoten auf der nächsttieferen Ebene. Gleichzeitig wandelte sich das Pluszeichen in ein Minuszeichen:



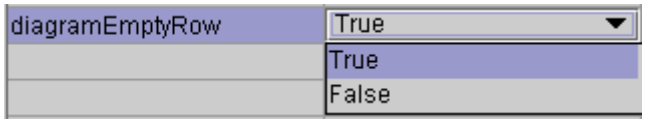
Eine Neuanlage von Knoten erfordert die Einordnung in die bestehende Hierarchie. Ein Knoten wird automatisch in die Hierarchie-Ebene eingeordnet, auf der er erzeugt wurde. Die dort gegebenenfalls. bereits vorhandenen Knoten werden hinten angestellt:



Der neue Knoten (gerahmtes Phantom) wird die Nummer 1.2 erhalten, der vorhandene Knoten mit der Nummer 1.2 wird automatisch auf Position 1.3 gerückt:

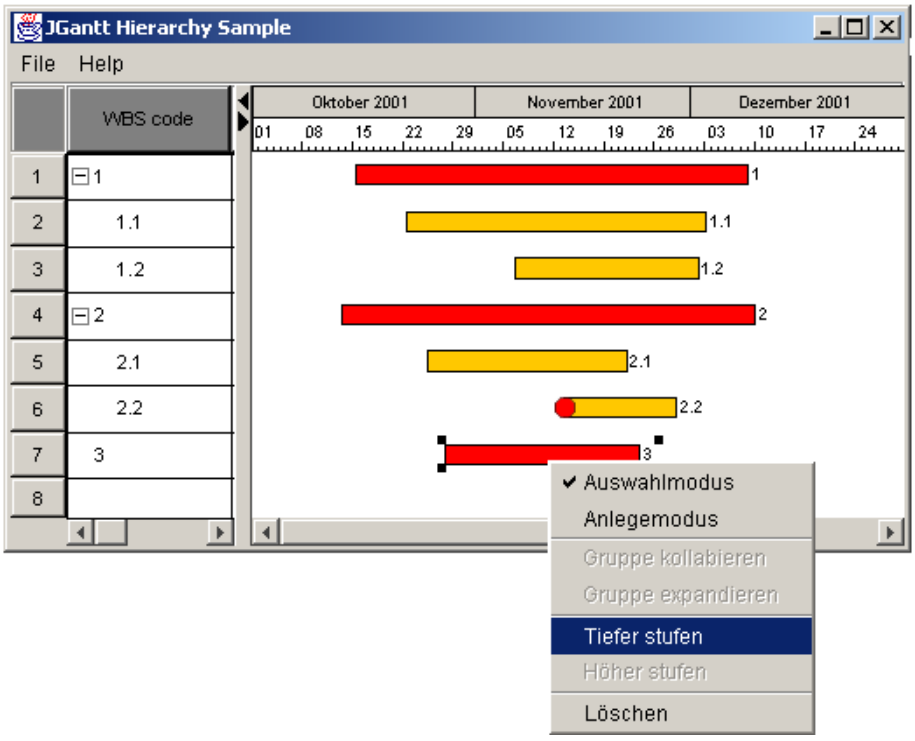


Das „hinten Anstellen“ erfolgt auch dann, wenn Sie den neuen Knoten nicht - wie gezeigt - links, sondern rechts von einem bestehenden Knoten einfügen. Wird ein Knoten in eine leere Zeile gesetzt, befindet er sich automatisch auf der obersten Ebene. Eine leere Zeile in einem Diagramm können Sie vorhalten, indem Sie die Eigenschaft **diagramEmptyRow** auf **True** setzen:

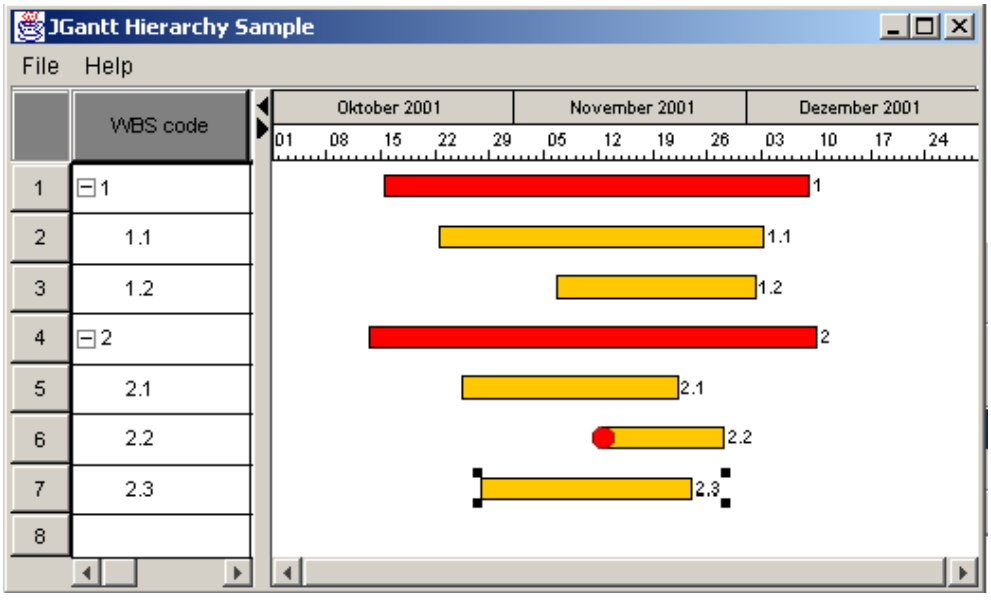



```
//Leere Zeile (letzte) im Diagramm vorhalten
jGantt1.setDiagramEmptyRow(true);
```

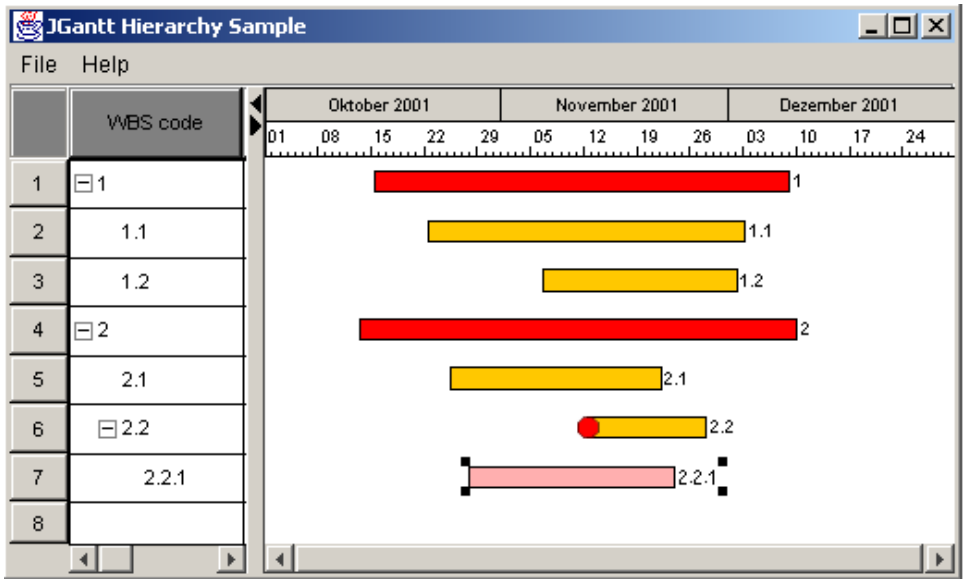
Dann befindet sich stets, wie oben abgebildet, am unteren Ende des Diagramms eine leere Zeile. Bei der Neuanlage eines Knotens in dieser Zeile entsteht automatisch eine neue leere Zeile. Einen Knoten, den Sie angelegt haben, können Sie innerhalb der Hierarchie höher oder tiefer stufen. Knoten auf der höchsten Ebene können nicht mehr höher gestuft werden, hingegen können Sie für Knoten auf der tiefsten Ebene eine noch tiefere Ebene erzeugen, indem Sie ihn tiefer stufen. Alle Knoten einer Ebene mit Ausnahme des ersten Knotens (d.h. der die Endnummer 1 besitzt) können tiefer gestuft werden. Umstufungen erfolgen über das Kontextmenü des Knotens, das Sie aufrufen, nachdem Sie einen Knoten markiert und die rechte Maustaste gedrückt haben. Im Beispiel unten wurde der Knoten 3 neu angelegt und markiert. Anschließend wurde das Kontextmenü aufgerufen:



Der Knoten wird nun beim tiefer Stufen auf den vorhandenen Vorläuferknoten bezogen und an die Reihe der bestehenden Knoten angehängt. Er erhält die Nummer 2.3:

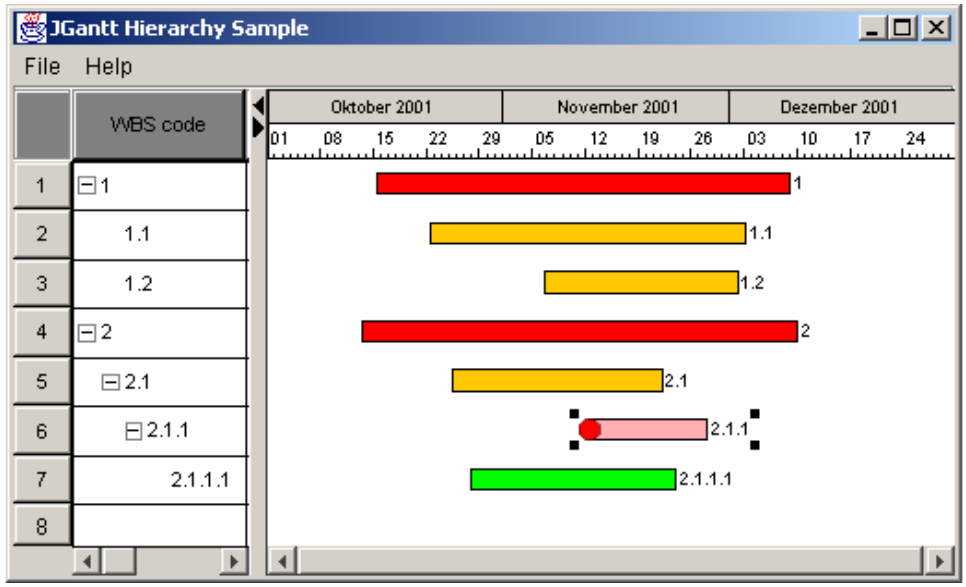


Beim erneuten Herunterstufen wird er wiederum auf den vorhandenen Vorläuferknoten bezogen und erhält er die Nummer 2.2.1:

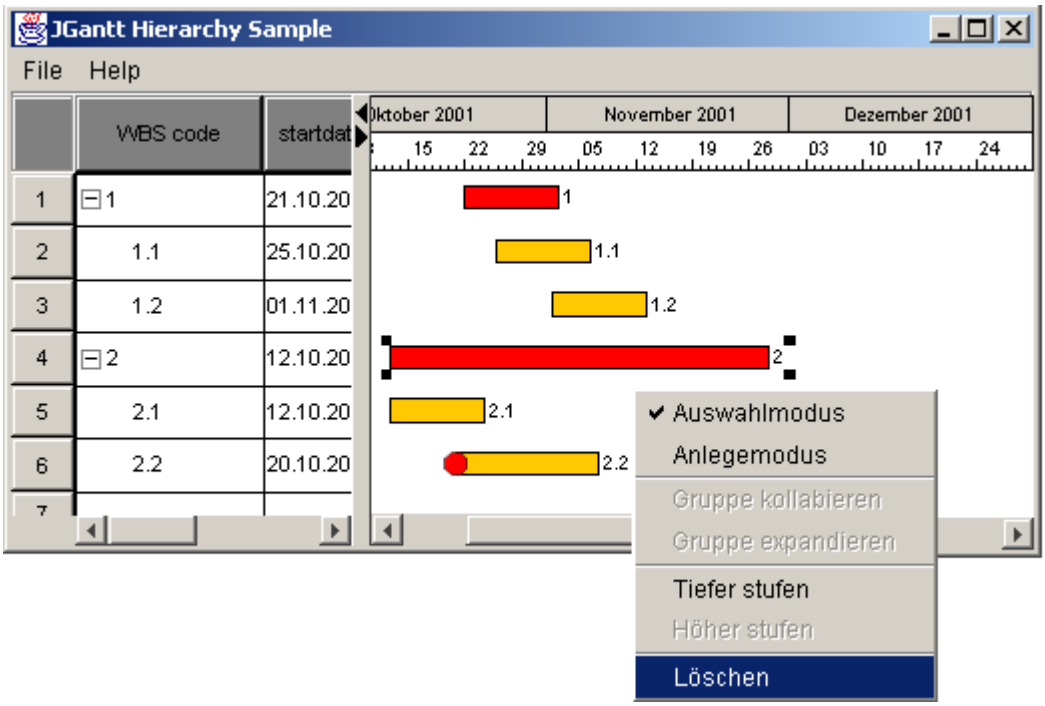


Wenn Sie denselben Knoten nun wieder zweimal höher stufen, durchläuft er dieselbe Abfolge in umgekehrter Reihenfolge und endet wieder als Knoten 3.

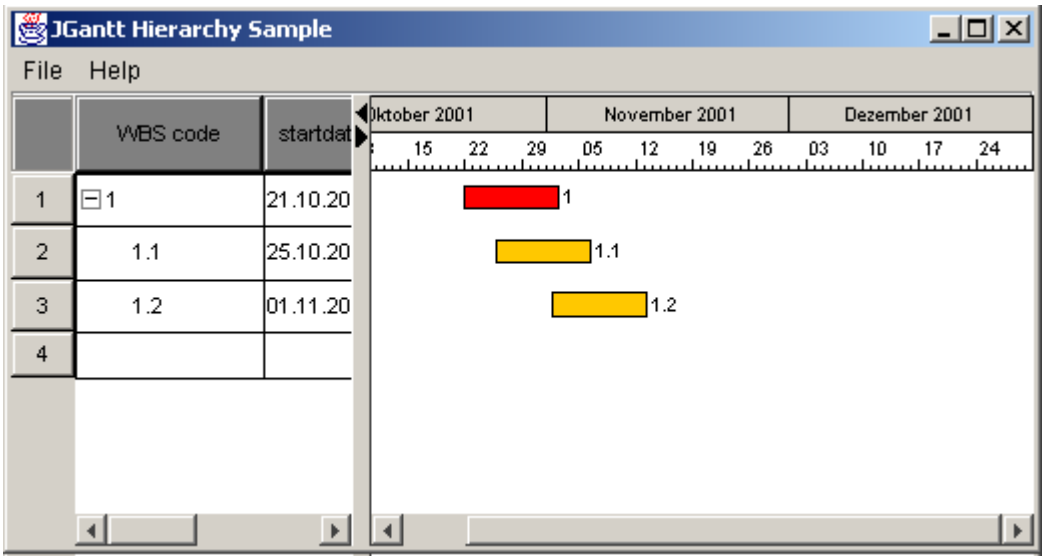
Im nächsten Beispiel wollen wir Ihnen zeigen, dass mit der Umstufung eines Knotens sich die Stufen der abhängigen Knoten ebenfalls ändern:



In diesem Beispiel wurde der Knoten 2.2 tiefer gestuft und wurde dadurch seinem Vorgänger, dem Knoten 2.1 untergeordnet. Er erhielt die Nummer 2.1.1. Sein abhängiger Knoten erhält entsprechend die Nummer 2.1.1.1. Beim Löschen eines Knotens werden seine abhängigen Knoten ebenfalls gelöscht. Im Beispiel unten soll Knoten 2, der zwei abhängige Knoten besitzt, gelöscht werden:



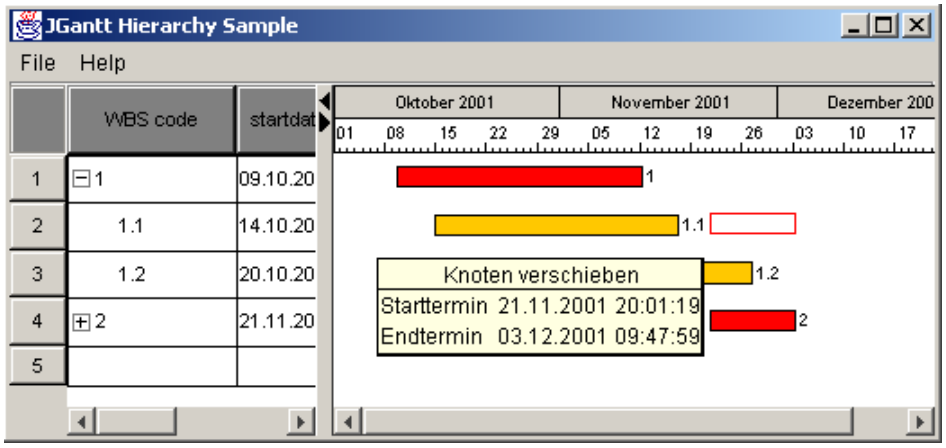
Das Ergebnis: der gesamte von Knoten 2 abhängige Teilbaum ist verschwunden.



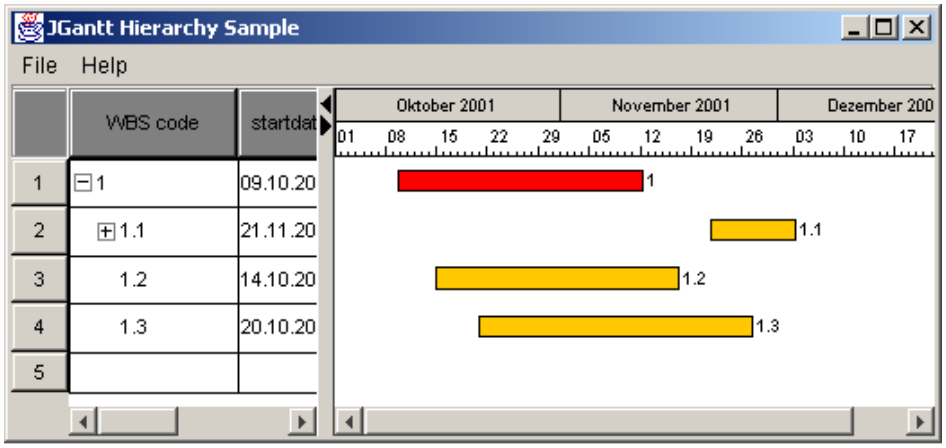
Beim Verschieben von Knoten unterscheidet sich das Verhalten für expandierte bzw. kollabierte Knoten.

Im Bild unten wird Knoten 2, angedeutet durch den Phantomrahmen, in die Zeile von Knoten 1.1 verschoben. In kollabiertem Zustand übernimmt der

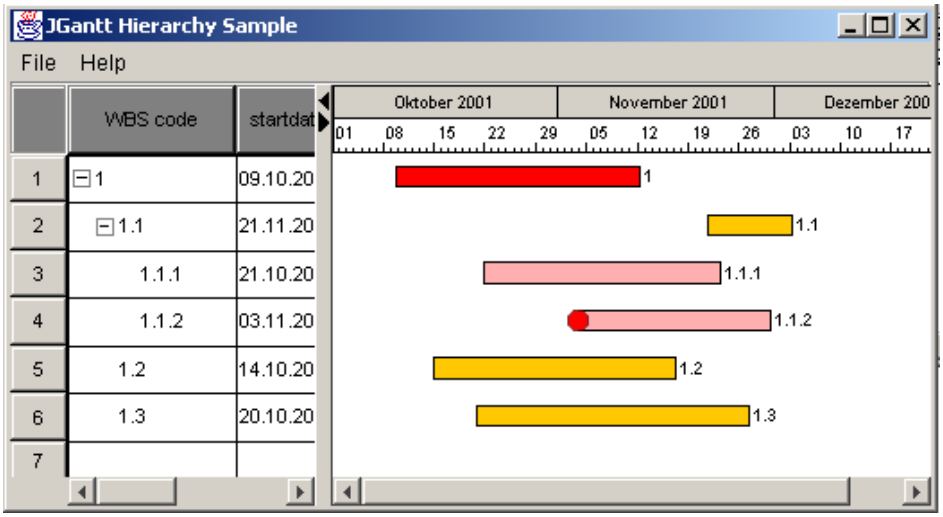
Knoten – ähnlich einer Neuanlage – den Rang, auf den er verschoben wurde. Ein dort vorhandener Knoten wird dahinter eingeordnet, alle folgenden Knoten rücken auf.



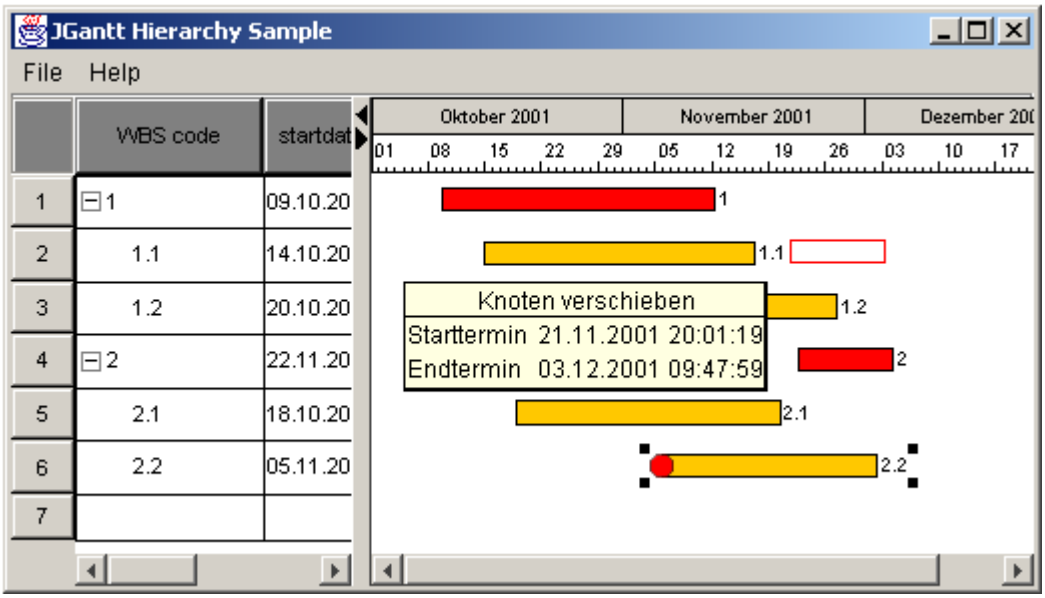
Knoten 2 hat also Rang 1.1 angenommen, der ehemalige Knoten 1.1 den Rang 1.2, der ehemalige Knoten 1.2 den Rang 1.3:



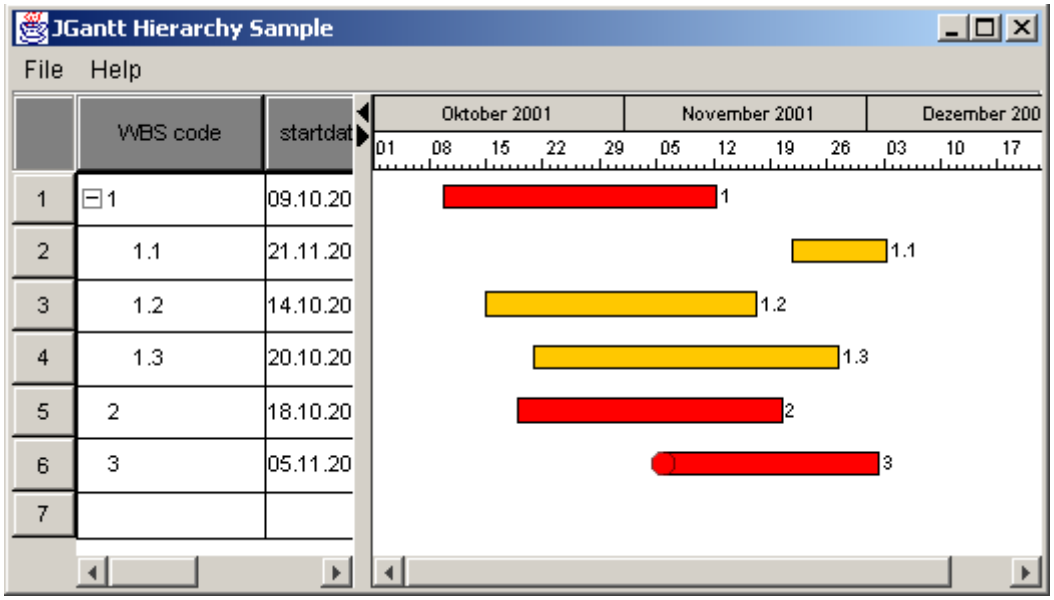
Knoten 1.1 ist weiterhin kollabiert, bei Expandierung zeigen seine untergeordneten Knoten eine entsprechend tiefere Einstufung:



Anders als ein kollabierter Knoten verhält sich ein expandierter Knoten. Er wird beim Verschieben von seinen abhängigen Knoten gelöst und wie beim Neuanlegen in die Hierarchie eingeordnet:



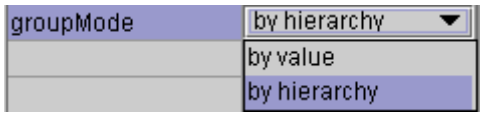
Im obigen Bild wurde Knoten 2, angedeutet durch den Phantomrahmen, in die Zeile von Knoten 1.1 verschoben. Er übernimmt nun den hierarchischen Platz von Knoten 1.1, während dieser auf 1.2 und Knoten 1.2 wiederum auf 1.3 verschoben wird. Gleichzeitig werden die ehemals von Knoten 2 abhängigen Knoten höher gestuft und erhalten die Ränge 2 und 3:



Wir haben Ihnen nun die wichtigsten Funktionalitäten und Interaktionen der Knotenhierarchie aus der Sicht des Anwenders gezeigt und können nun zur Programmierung fortschreiten.

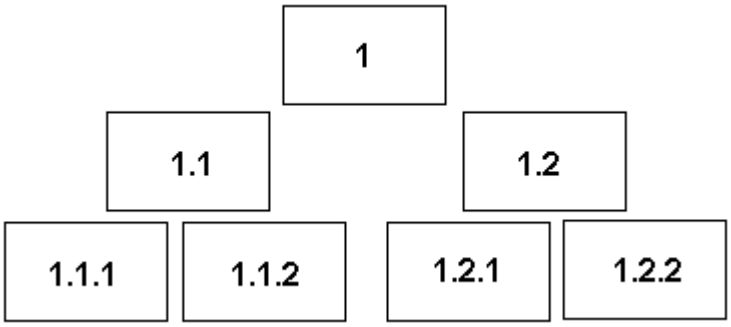
Hierarchie aus der Sicht des Programmierers

Die Programmierung der Hierarchie ist einfach und im Wesentlichen darauf beschränkt, die hierarchische Struktur einzurichten und dem JGantt bekannt zu machen. Die Interaktionen des Benutzers werden von VARCHART JGantt automatisch verwaltet. Als erstes legen Sie mit der Eigenschaft groupMode fest, dass Sie eine Hierarchie einrichten wollen:




```
// Einrichten der Hierarchie
jGantt1.setGroupMode(de.netronic.jgantt.JGantt.GROUP_BY_HIERARCHY);
```

Um die Hierarchie zunächst auf logischer Ebene einzurichten, legen Sie im Entitäten-Set ein Attribut an und benennen es, z.B. mit dem Namen „wbs“ (für Work Breakdown Structure). Anschließend füllen Sie es mit Hierarchie-Code. Der Hierarchie-Code bezeichnet die Position, die der Knoten innerhalb der Hierarchie einnehmen soll:



usw.

Der Hierarchie-Code besteht aus Ziffern und Punkten. Die Ziffern bezeichnen die Positionen innerhalb einer Matrix, wobei für jede eingefügte Ebene eine Ziffer, getrennt durch einen Punkt, hinzugefügt wird. Innerhalb einer Ebene wird pro Knoten fortlaufend und aufsteigend gezählt. Der Punkt bildet das festgelegte Trennsymbol zwischen den Ziffern. Die Reihenfolge wird numerisch, nicht alphabetisch interpretiert. Dies bedeutet, dass z.B. 1.2.3 vor 1.12.3 angeordnet wird und führende Nullen ignoriert werden. Der Hierarchie-Code muss nicht eindeutig und kann lückenhaft sein, er wird in jedem Fall interpretiert und an zweifelhaften Stellen automatisch vom System geändert.

Nachdem Sie die Hierarchie logisch definiert und das Hierarchie-Attribut mit Daten gefüllt haben, wird es dem JGantt-Objekt bekannt gemacht. Die entsprechende Option können Sie im Design-Mode einstellen...

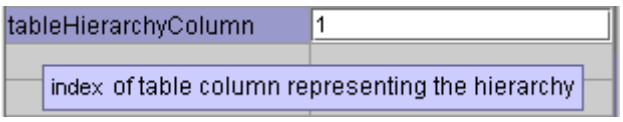
... oder im Quell-Code programmieren:

```
// Hierarchie-Attribut bekannt machen
jGantt1.setGroupHierarchyBy("wbs");
```

Für manche Anwendungen kann es sinnvoll sein, den Hierarchie-Code in einer Tabellenspalte darzustellen. Dies erfolgt - wie bei anderen Attributen - über die Definition der Tabellenspalten:

```
// Tabellenspalten definieren
jGantt1.setTableColumns(new String[]{"wbs", "start", "end");
```

Die Tabelle zeigt nach dieser Definition drei Spalten: eine mit dem Hierarchie-Code, eine mit dem Anfangsdatum und eine mit dem Enddatum. Das Sichtbarmachen des Hierarchie-Codes in der Tabelle bedeutet allerdings nicht automatisch, dass hier auch die Einrückung erscheint. Sie können die Einrückung auf jede beliebige Tabellenspalte setzen. Dazu geben Sie dem JGantt-Objekt den Index der gewünschten Tabellenspalte bekannt. Wenn dieser auf 0 steht, ist keiner Spalte die Einrückung zugewiesen, sobald Sie eine positive Ziffer setzen, wird in der entsprechenden Tabellenspalte die Einrückung sichtbar:




```
// Index der Tabellenspalte setzen, die die Hierarchie anzeigen soll
jGantt1.setTableHierarchyColumn(1);
```

Mit dem Index 1 wird die hierarchische Einrückung auf die erste Spalte gesetzt, die den Inhalt des Attributs **wbs**, also den hierarchischen Code darstellt:

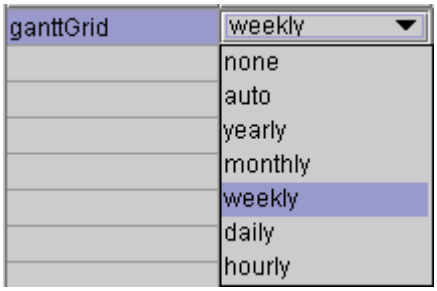
	WBS code	startdate	enddate
1	⊟1	04.11.2001	31.12.2001
2	1.1	06.11.2001	16.12.2001
3	1.2	27.11.2001	30.12.2001
4	⊟2	27.10.2001	19.12.2001
5	2.1	09.11.2001	21.11.2001
6	2.2	20.11.2001	02.12.2001
7			

Damit haben Sie nun die wesentlichen Eigenschaften für die Anwendung und das Programmieren der hierarchischen Darstellung von Knoten erfasst. Wir können damit Organisation und Darstellung von Knoten hinter uns lassen und uns der Visualisierung zeitlicher Muster im Gantt-Diagramm zuwenden.

5.4 Visualisierung zeitlicher Muster

In der Zeitskala und im Gantt-Graphen eines VARCHART JGantt-Diagramms können Sie ausgewählte Zeitbereiche durch farbige Linien und Streifen hervorheben. Dadurch wird die Einordnung der Vorgänge in ihr zeitliches Gefüge auf einen Blick erkennbar.

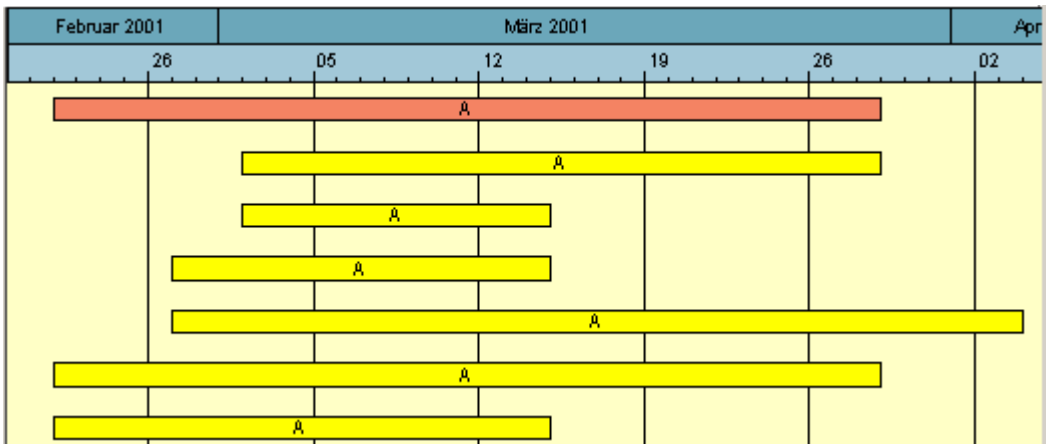
Eine zusammengehörige Menge von Linien oder Streifen wird Gitter genannt. Grundsätzlich gibt es in VARCHART JGantt zwei Typen von Gittern: das einfache Gitter (ganttGrid) und das Kalendergitter (ganttCalendarGrid). Das erstere ist ein Liniengitter, das den Hintergrund des Gantt-Graphen mit periodisch auftretenden Linien bedeckt, z.B. um das Ende einer jeden Woche zu markieren. Es wird durch eine einfache Methode gesetzt, wobei die Periodizität in vorgegebenen Schritten (stündlich, täglich, wöchentlich usw.) moduliert werden kann. Die Methode können Sie im Design-Modus einstellen:



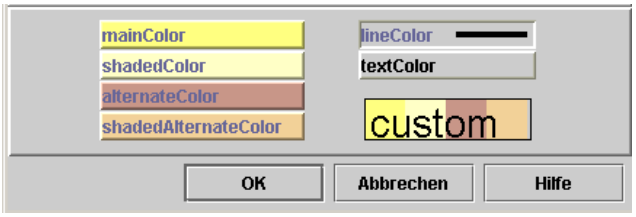
...oder im Quellcode direkt schreiben:

```
// einfaches Liniengitter setzen, Periodizität wöchentlich
jGantt1.setGanttGrid(de.netronic.jgantt.JGantt.GANTT_GRID_WEEKLY);
```

Das Ergebnis sieht so aus:



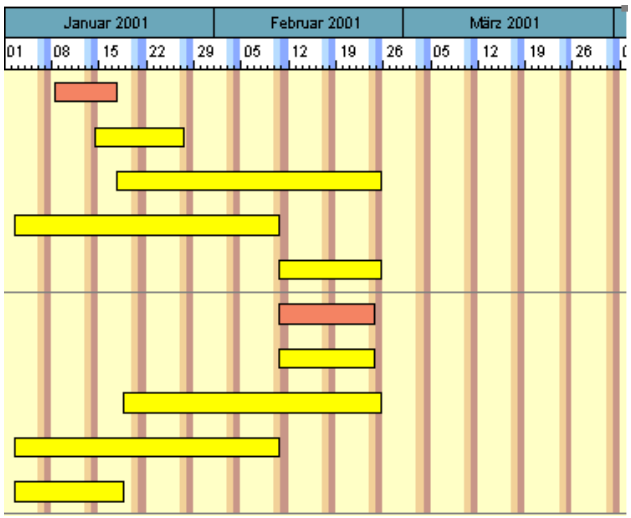
Einfache schwarze Linien trennen die Wochen. Die Farbe des Liniengitters wird durch die Linienfarbe (line color) des Gantt-Farbschemas (ganttColor-Scheme) bestimmt, die in unserem Beispiel auf custom eingestellt und schwarz ist:



Sie können die Gitterfarbe durch das Wechseln der Linienfarbe verändern. Während Sie mit einem einfachen Liniengitter nur einfarbige Linien und unkomplizierte zeitliche Muster darstellen können, sind Sie mithilfe des Kalendergitters in der Lage, komplexe zeitliche Muster abzubilden. Das Kalendergitter wird von einem sogenannten Kalender-Objekt verwaltet. In diesem definieren Sie wechselnde Zeitabschnitte aus Arbeits- und Nichtarbeitszeiten mithilfe von Profilen, um mit ihnen andere Objekte des Diagramms bei der Visualisierung der zeitlichen Muster zu unterstützen.

Im Gegensatz zu einem Liniengitter kann ein Kalendergitter das Muster von Profilen abbilden. Daraus ergeben sich zwei grundlegende Unterschiede im Erscheinungsbild der beiden Gitter: Das Kalendergitter kann alle Arten regelmäßiger und unregelmäßiger Zeitmuster abbilden, und die eingefärbten Streifen können beliebig breit sein.

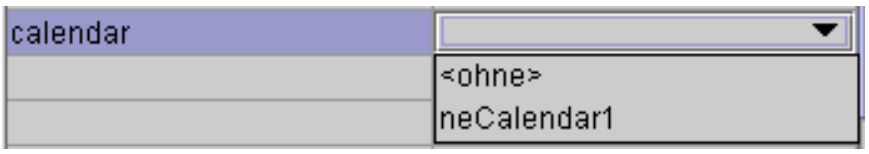
Im Gantt-Graphen gibt es neben dem Kalendergitter noch andere Objekte, die die durch ein Profil vordefinierten Zeitmuster visualisieren können, z.B. die Zeitskala:



Wir werden uns in den folgenden Kapiteln mit der Visualisierung komplexer Zeitmuster über Kalendergitter und die Zeitskala befassen.

5.4.1 Kalender

Ein Kalender bildet das logische Rahmenobjekt, innerhalb dessen die für die Visualisierung von zeitlichen Mustern nötigen Objekte verwaltet werden. Ein Kalender entsteht automatisch beim Anlegen eines JGantt-Objekts. Jedem JGantt-Objekt ist genau ein Kalender zugeordnet. Ein JGantt-Objekt kann seinen Kalender wechseln, was Sinn macht, wenn Sie z.B. mit mehreren JGantt-Objekten arbeiten und allen denselben Kalender zuweisen wollen, damit alle dieselben Zeitmuster aufweisen. Da das Wechseln eines Kalenders dennoch relativ selten benötigt wird, ist es eine versteckte Eigenschaft des JGantt-Objektes, die Sie im Design-Modus folgendermaßen einstellen können:



Die Option **ohne** bedeutet, dass kein fremder Kalender zugewiesen wurde und somit der Gantt-eigene Standard-Kalender gesetzt ist. Andere Kalender werden angezeigt, falls Sie diese im Source-Code Ihres Programms erzeugt haben. Hier die Anlage und Zuweisung im Source-Code:

```
//Import der notwendigen Klasse
import de.netronic.bean.calendar.*;

// Anlage einer neuen Kalender-Instanz
NeICalendar neCalendar1 = new NeCalendar();

//Definiton des Kalenders
...


//Zuweisung des Kalenders
jGantt1.setCalendar(neCalendar1);
```

Die Definition des Kalenders erfolgt durch die Einrichtung von so genannten Profilen.

5.4.2 Profile

In einem Kalender können verschiedene Pakete mit Zeitmustern vorhanden sein. Ein solches Paket heißt **Profil**. Profile besitzen Namen, über die sie identifizierbar sind. Die Profile eines Kalenders können über ihre Namen verschiedenen Objekten im Gantt-Diagramm zugewiesen werden. Die Voraussetzung dafür ist, dass die Profile vorher angelegt, benannt und bestückt wurden.

```
//Anlage eines Profils
 profile = neCalendar1.createProfile();

 //Benennung des Profils
 profile.setName("CalendarProfile");

 //Bestückung des Profils
 ...
```

Mit der Bestückung von Profilen befassen wir uns später, wir setzen für den Augenblick einmal das Vorhandensein bestückter Profile voraus.

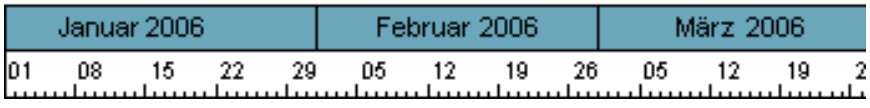
Die Visualisierung eines Profils in der Zeitskala

Hier die Zuweisung eines Profils zur Zeitskala im Design-Mode:

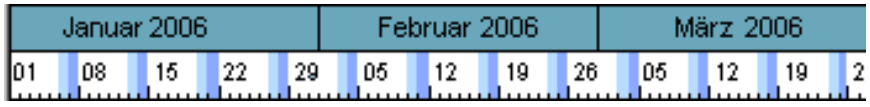
...und im Source-Code:

```
//Zuweisung eines Profils zur Zeitskala
jGantt1.setTimeScaleDisplayProfile("CalendarProfile");
```

Die Zuweisung eines Profils zur Zeitskala ist direkt mit der entsprechenden Visualisierung im Zeitskalenband verbunden:



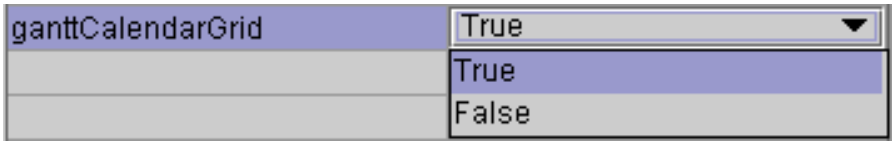
Zeitskala vor der Zuweisung



Zeitskala nach der Zuweisung

Die Visualisierung eines Profils im Gantt-Graphen

Für die Visualisierung von Profilen als Hintergrundmuster des Gantt-Graphen ist grundsätzlich das Einschalten des Kalendergitters notwendig. Dazu setzen Sie dieses auf **True**:

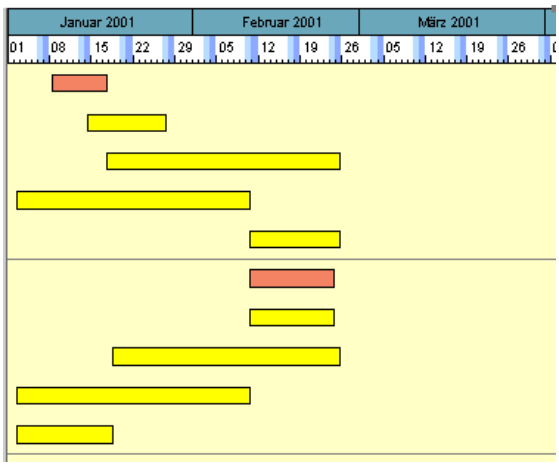


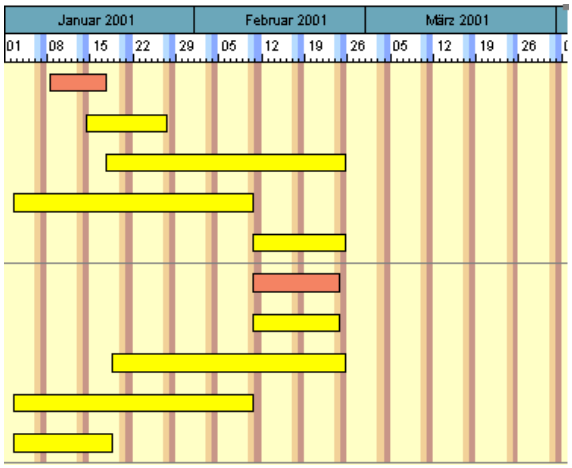

```
//Setzen des Kalendergitters
jGantt1.setGanttCalendarGrid(true);
```

Bevor Sie jedoch das Kalendergitter sehen können, müssen Sie definieren, ob das zugrunde liegende Profil für den gesamten Gantt-Graphen gültig sein soll oder ob verschiedene Profile für Teilbereiche wirksam werden sollen. Nehmen wir zunächst den einfachen Fall: Ein einziges Profil soll für das gesamte Kalendergitter des Gantt-Graphen gültig sein, d.h. es soll überall das gleiche Zeitmuster zeigen. Dazu setzen Sie das **ganttDisplayProfile**:

```
//Zuweisung eines Profils zum Gantt-Graphen
jGantt1.setGanttDisplayProfile("CalendarProfile");
```

Hier das Ergebnis:





Das dem Gantt-Graphen zugewiesene Profil ist im rechten Diagramm durch das Muster aus senkrechten Streifen sichtbar geworden. Da dasselbe Profil im vorigen Kapitel auch der Zeitskala zugewiesen worden war, zeigen Zeitskala

und Kalendergitter im obigen Bild dasselbe zeitliche Muster. Es kann aufgrund der verschiedenen Farbschemata von Zeitskala und Kalendergitter grafisch unterschiedlich ausgestaltet werden. Bevor wir nun den Fall betrachten, für verschiedene Knotengruppen unterschiedliche Profile im Gantt-Graphen darzustellen, zunächst ein paar grundsätzliche Anmerkungen zur Funktionsweise eines Knotenprofils.

GanttDisplayProfile und Knotenprofil

Ein Knotenprofil weisen Sie über die Option nodeProfile allen Knoten des Diagramms zu:

```
//Zuweisung eines Profils zu den Knoten
 jGantt1.setNodeProfile("CalendarProfile");
```

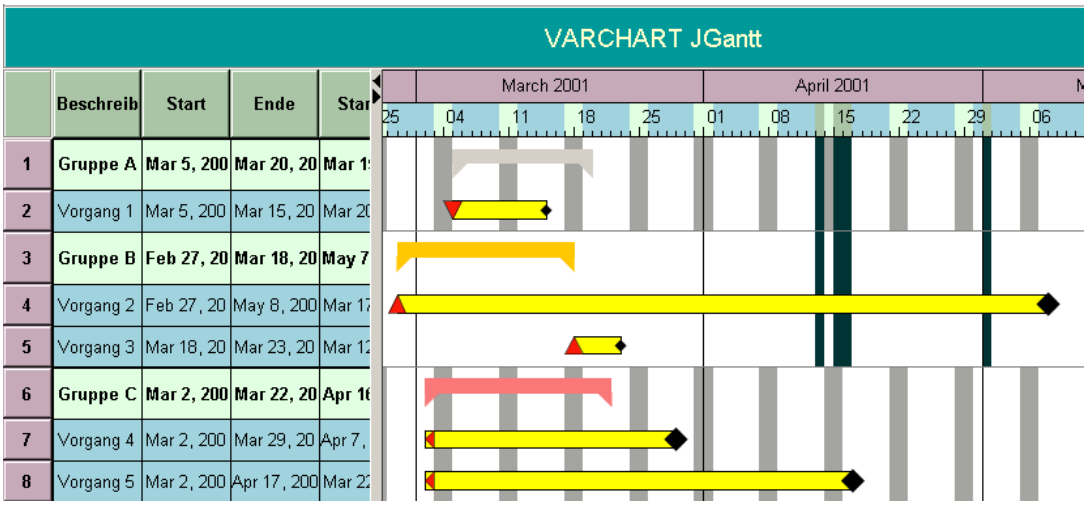
Das Knotenprofil kann dasselbe oder ein anderes sein als das **ganttDisplayProfile**. Neben der Darstellung des Kalendergitters dienen Knotenprofile als Grundlage für zeitliche Berechnungen und für das Sprungverhalten von Knoten beim Verschieben. Das **ganttDisplayProfile** hingegen dient ausschließlich der Darstellung des Kalendergitters. Wenn zeitliche Berechnungen und Anzeige übereinstimmen sollen, ist es sinnvoll, beide Profile gleich oder nur das Knotenprofil zu setzen. Sollen sich Berechnungen und Anzeige jedoch unterscheiden, indem z.B. nur Wochenenden, aber keine Feiertage dargestellt werden sollen, die Feiertage aber mitberechnet werden sollen, dann sollten Sie entsprechend unterschiedliche Profile definieren und das eine dem Knotenprofil und das andere dem **ganttDisplayProfile** zuweisen. Ein **ganttDisplayProfile** muss nicht zugewiesen werden, wenn Sie ein Knotenprofil setzen. Wenn kein **ganttDisplayProfile** zugewiesen wurde, wird zur Visualisierung das Knotenprofil verwendet.

Knotenspezifische Profile

Sie können über die Option nodeProfileBy einzelnen Knoten und Gruppenknoten ein Profil zuweisen:

```
//Zuweisung eines Profils zu den Knoten
jGantt1.setNodeProfileBy("profileAttr");
```

Hierbei wird der Name des Profils einem Attribut der Knotenentität entnommen (das im obigen Beispiel den Namen **profileAttr** trägt), den Sie diesem Attribut für jeden Knoten individuell zuweisen können. Mit diesem Aufruf können Sie das Kalendergitter z.B. für Knotengruppen gestalten, wie im Bild unten. Es zeigt drei verschiedene Profile in seinem Kalendergitter. Für die obere Knotengruppe werden Wochenenden und Feiertage dargestellt (regelmäßige graue und unregelmäßige schwarze Streifen), für die mittlere Knotengruppe nur Feiertage und für die untere nur Wochenenden:



Bevor wir jedoch die Details dieses Diagramms erläutern, ist es sinnvoll, sich erst einmal mit der Bestückung von Profilen zu befassen.

5.4.3 Bestückung von Profilen

Durch die Bestückung von Profilen mit Zeitspannen legen Sie das zeitliche Muster fest, das von den Objekten des Gantt-Diagramms dargestellt werden soll. Zeitspannen zeichnen sich durch verschiedene Qualitäten aus.

Zum einen können sie entweder als Arbeitszeit oder als Nichtarbeitszeit definiert sein, zum anderen können sie entweder einmalig sein, wie z.B. ein Urlaubstag, oder periodisch auftreten (wiederkehrend), wie z.B. die Wochenenden.

Zeitspannen können Sie auf verschiedene Arten und Weisen erzeugen. Entweder legen Sie über die Kalender-Methode createTimeSpan () eine leere Zeitspanne an und füllen sie anschließend über setSpan (String newValue). Oder Sie legen gleich bei der Anlage sämtliche Eigenschaften der Zeitspanne

über drei Parameter fest: die Spanne, die Aktivität und die Identifikationsnummer.

```
createTimeSpan(string span, int activity, int id);
```

Durch das Format des Parameters **span** bietet VARCHART JGantt eine Vielzahl von bequemen Möglichkeiten, bestimmte Eigenschaften der Zeitspanne festzulegen: Falls die Reihenfolge des Datums dem Schema 22.5.2014 7:30-25.5.2014 17:00 entspricht, ist die Zeitspanne einmalig. Falls die Reihenfolge des Datums dem Schema 22.5.2014-25.5.2014 7:30-17:00 entspricht, ist die Zeitspanne im spezifizierten Zeitraum täglich wiederkehrend. Das Tagesdatum kann auch durch definierte Namen ersetzt werden wie:

Montag ..Sonntag (alle Montage, alle Dienstage usw.)

Erster Montag .. Erster Sonntag (jeden ersten Montag im Monat usw.)

Letzter Montag ..Letzter Sonntag (jeden letzten Montag im Monat usw.)

* (jeder Tag)

Falls Sie die oberen drei definierten Tage verwenden, wird die Zeitspanne automatisch wiederkehrend. Den Asterisk verwenden Sie als Platzhalter für alle Elemente einer Kategorie, z.B. alle Monate eines Jahres oder alle Jahre innerhalb des gesamten Kalenders. Weiterhin können Sie definierte Namensstrings von beweglichen Feiertagen (z.B. „Karfreitag“) verwenden. Die Details dazu finden Sie im Referenzteil des Handbuchs.

Über den Parameter **activity** bestimmen Sie, ob es sich bei der Zeitspanne um eine Arbeitszeit oder eine Nichtarbeitszeit handelt. Die Ziffer 0 definiert eine Zeitspanne als Nichtarbeitszeit. Die Werte zwischen 1 und 1.000 definieren die Zeitspanne als Arbeitszeit und geben einen Effizienzwert an, zum Beispiel die Menge an Arbeitskraft, die während der Zeitspanne eingesetzt wurde. Dabei repräsentiert 1.000 die vollwertige Arbeitszeit, 1 den kleinstmöglichen Anteil.

Der letzte Parameter **Id**, ist eine Identifikationsnummer, die Sie vergeben können, um - in einem anderen Teil Ihres Programms - bestimmte Zeitspannen farblich hervorzuheben. Die dazugehörige **Dynamische Farbgebung** wird in einem der nachfolgenden Kapitel beschrieben.

Definieren wir zunächst einmal mit dem bisherigen Wissen die Zeitspannen, die das zeitliche Muster eines Kalendergitters bilden sollen. Die Aufgabe ist,

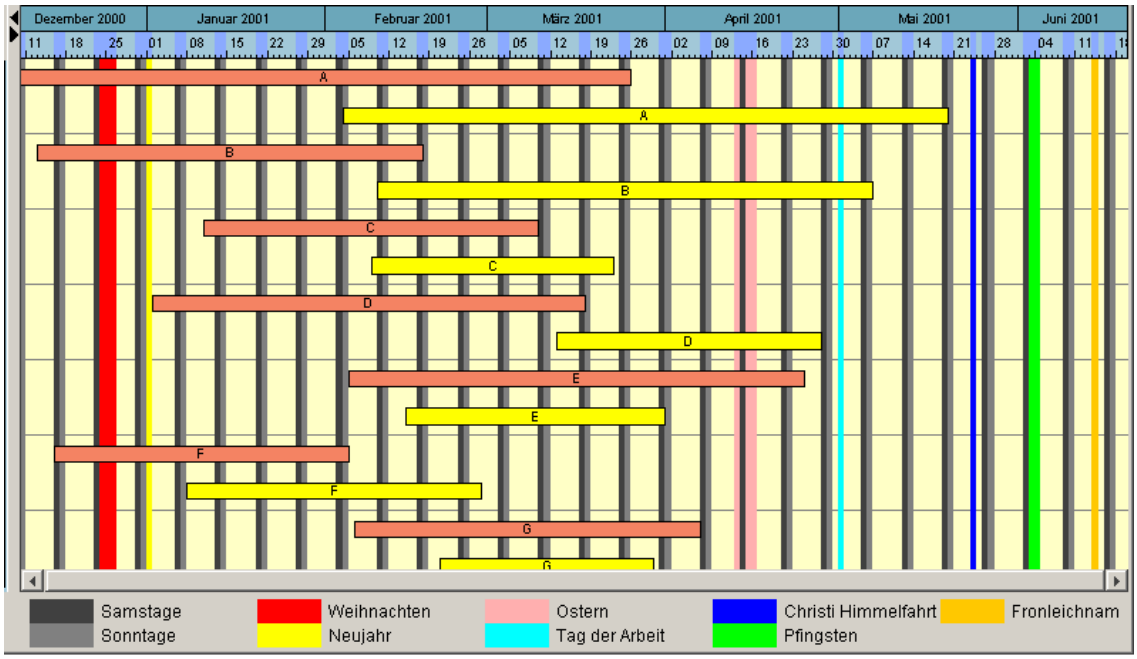
die arbeitsfreien Wochenenden und Feiertage eines Jahres zu definieren. Alle Wochenenden sollen in Grautönen, d.h. für den Samstag hellgrau, für den Sonntag dunkelgrau dargestellt werden, Feiertage hingegen jeweils eine eigene Farbe erhalten. Daher müssen nun alle angelegten Zeitspannen Nichtarbeitszeiten (activity = 0) sein und 10 verschiedenen Farbgruppen (Id= 1..10) zugeordnet werden:

```
//Definition der Wochenenden

//span: Jeden Samstag in jedem Monat jedes Jahres
ts0 = neCalendar1.createTimeSpan("Samstag.*.*", 0, 1);

//span: Jeden Sonntag in jedem Monat jedes Jahres
ts1 = neCalendar1.createTimeSpan("Sonntag.*.*", 0, 2);

// Definition der Feiertage
//span: Silvester jedes Jahres
ts2 = neCalendar1.createTimeSpan("1.1.*", 0, 3);

//span: Karfreitag und Ostern jedes Jahres
ts3 = neCalendar1.createTimeSpan("Karfreitag.*.*", 0, 4);
ts4 = neCalendar1.createTimeSpan("Ostersonntag.*.*", 0, 4);
ts5 = neCalendar1.createTimeSpan("Ostermontag.*.*", 0, 4);

//span: Maifeiertag jedes Jahres
ts6 = neCalendar1.createTimeSpan("1.5.*", 0, 5);

//span: Christi Himmelfahrt jedes Jahres
ts7 = neCalendar1.createTimeSpan("ChristiHimmelfahrt.*.*", 0, 6);

//span: Pfingsten jedes Jahres
ts8 = neCalendar1.createTimeSpan("Pfingstsonntag.*.*", 0, 7);
ts9 = neCalendar1.createTimeSpan("Pfingstmontag.*.*", 0, 7);

//span: Fronleichnam jedes Jahres
ts10 = neCalendar1.createTimeSpan("Fronleichnam.*.*", 0, 8);

//span: Allerheiligen
ts11 = neCalendar1.createTimeSpan("1.11.*", 0, 9);

//span: Weihnachten jedes Jahres
ts12 = neCalendar1.createTimeSpan("24.12.*", 0, 10);
ts13 = neCalendar1.createTimeSpan("25.12.*", 0, 10);
ts14 = neCalendar1.createTimeSpan("26.12.*", 0, 10);
```

Nun können Sie die definierten Zeitspannen einem oder mehreren Profilen zuweisen. Für das im nächsten Kapitel behandelte Beispiel der dynamischen Farbgebung ist es sinnvoll, ein einziges Profil zu verwenden:

```
profile.addTimeSpan(ts0);
…
profile.addTimeSpan(ts14);
```

Grundsätzlich kann die Farbgebung statisch oder dynamisch erfolgen. Die statische Farbgebung führen Sie über die Zuweisung des JGantt-Farbschemas zum JGantt-Objekt durch. Die Folge wäre, dass das Zeitmuster des Liniengitters maximal zwei unterschiedliche Farben zeigen kann. Bei der dynamischen Farbgebung ist die Farbe datengesteuert, so dass jede Linie oder Liniengruppe eine eigene Farbe zeigen kann. Die steuernden Daten bei den Kalendergittern sind die Ids der Zeitspannen, von denen wir 10 verschiedene vergeben haben, so dass 10 verschiedene Farben im Gitter abgebildet werden:



Das Ergebnis haben wir in dem obenstehenden Bild schon vorweggenommen. Jede Zeitspanne gehört einer eigenen Farbgruppe an. So sind alle Samstage hellgrau, alle Sonntage dunkelgrau, die Weihnachtstage rot, die Ostertage rosa usw. eingefärbt. Wie die dynamische Farbgebung im Detail funktioniert, erfahren Sie im Kapitel über dynamische Farben.

Zunächst wollen wir Ihnen noch zeigen, wie Sie gruppenspezifische Kalendergitter einrichten. Dazu verteilen Sie die definierten Zeitspannen auf verschiedene Profile und weisen diese den Attributen bestimmter Knoten zu. Im untenstehenden Quellcode wird ein Profil mit dem Namen **all** angelegt und mit Wochenenden und Feiertagen bestückt. Anschließend folgen Profil2 („weekend“) nur mit den Wochenenden und Profil3 („holidays“) nur mit den Feiertagen:

```
//Anlage und Bestückung von Profil1 mit Wochenenden und Feiertagen
profile1 = theCalendar.createProfile();
profile1.setName("all");
```

```
profile1.addTimeSpan(ts0);
profile1.addTimeSpan(ts1);
…
profile1.addTimeSpan(ts13);
profile1.addTimeSpan(ts14);

//Anlage und Bestückung von Profil2 nur mit Wochenenden
profile2 = theCalendar.createProfile();
profile2.setName("weekend");
profile2.addTimeSpan(ts0);
profile2.addTimeSpan(ts1);

//Anlage und Bestückung von Profil3 nur mit Feiertagen
profile3 = theCalendar.createProfile();
profile3.setName("holiday");
profile3.addTimeSpan(ts2);
profile3.addTimeSpan(ts3);
…
profile3.addTimeSpan(ts13);
profile3.addTimeSpan(ts14);
```

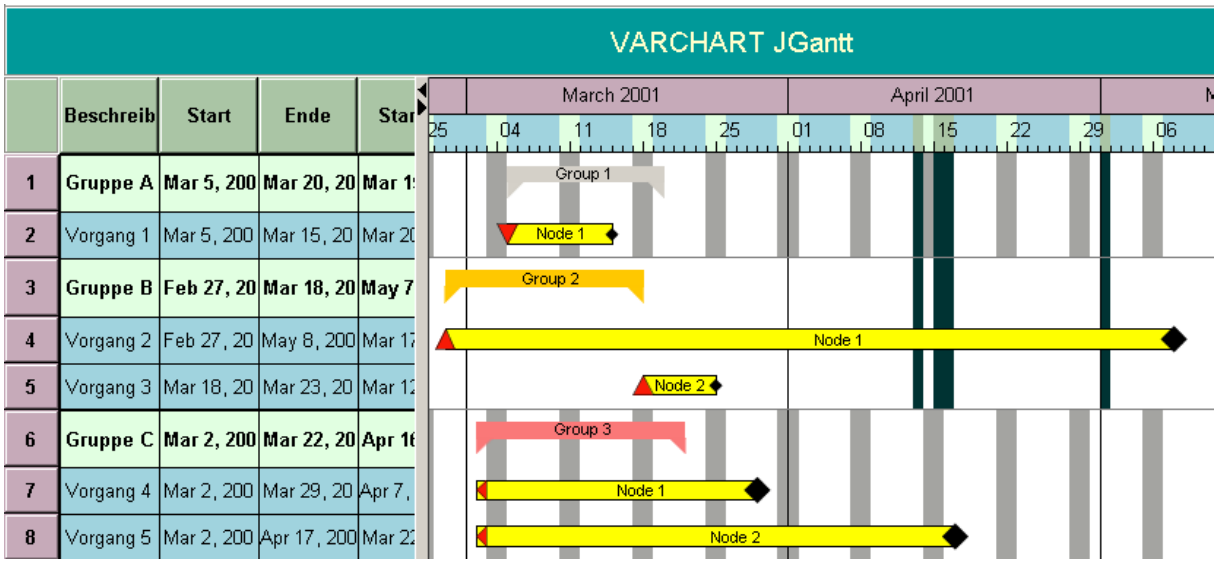
Anschließend werden die Profile „holiday“ und „weekend“ dem entsprechenden Attribut einer Knoten-Entität und einer Gruppenknoten-Entität zugewiesen:

```
// Zuweisung des Profils "holiday" zum Attribut "profile" in den
// Entitäten nodeEntity1 und groupEntity1
group2Entity[1].setValue("profile","holiday");
node2Entity[1].setValue("profile","holiday");
node2Entity[2].setValue("profile","holiday");


// Zuweisung des Profils "weekend" zum Attribut "profile" der
// Entitäten nodeEntity2 und groupEntity2
group3Entity[1].setValue("profile","weekend");
node3Entity[1].setValue("profile","weekend");
node3Entity[2].setValue("profile","weekend");
```

Nun wird über **setNodeProfile** das Default-Profil **all** gesetzt, das immer dann von einem Knoten angezeigt wird, wenn in sein Attribut kein Profil eingetragen wurde. Anschließend werden über **setNodeProfileBy** die individuellen Knotenprofile gesetzt, die dem Attribut „profile“ entnommen werden:

```
//Setzen des Default-Profils für Knoten und Gruppenknoten
jGantt1.setNodeProfile("all");
jGantt1.setGroupNodeProfile("all");

//Setzen des individuellen Profils für Knoten und Gruppenknoten
jGantt1.setNodeProfileBy("profile");
jGantt1.setGroupNodeProfileBy("profile");
```

Im untenstehenden Bild sehen Sie nun das Default-Profil in der oberen Knotengruppe, die Wochenenden und Feiertage anzeigt. Die Knoten dieser Gruppe besitzen keinen Eintrag im Attribut **Profile**. Darunter befindet sich eine Knotengruppe, die das Profil **holiday** anzeigt, während in der untersten Gruppe das Profil **weekend** erscheint.



Damit ist die Gestaltung eines Kalenders für die Darstellung von komplexen Zeitmustern zunächst abgeschossen.

5.5 Kalenderpersistenz

Bevor wir uns der Farbgebung zuwenden, ein paar Erläuterungen zum Abspeichern und Laden des Kalenders. Um den Kalender persistent zu machen, ist es erforderlich, ihn mit der AppData zu koppeln, was mit dem folgenden Aufruf möglich ist:

```
theCalendar.setAppData (AppData);
```

Zugreifen auf den Kalender können Sie über drei verschiedene Entitäten-Sets: eines für die Profile, eines für die Zeitspannen und eines für die Beziehungen zwischen Profilen und Zeitspannen. Die genannten Entitäten-Sets besitzen entsprechende Standard-Namen: **Profiles**, **TimeSpans**, **ProfileTimeSpanRelations**. Die Namen können geändert werden. Mit diesen Entitäten-Sets sind die Kalenderdaten vollständig beschrieben. Bei der Zuwei-

sung werden die Kalenderdaten in die AppData geschrieben und die Daten aus den drei Entäten-Sets in den Kalender eingelesen. Nach der Kopplung speichern Sie die AppData mit den Kalenderdaten:

```
AppData.saveXML("theDataAndCalendarFile.xml",
                  calendar.fetchEntitySetnames(), false);
```

Den saveScheme-Parameter sollten Sie nur in Ausnahmefällen auf **true** setzten. Beim Wiedereinlesen der Kalenderdaten haben Sie die Möglichkeit, diese mit oder ohne die Entitäten-Sets der Profile, der Zeitspannen und Beziehungen einzulesen:

```
//Mit Entitäten-Sets
AppData.loadXML("theDataAndCalendarFile.xml",
                           calendar.fetchEntitySetnames());

//Ohne Entitäten-Sets
AppData.loadXML("theDataAndCalendarFile.xml");
```

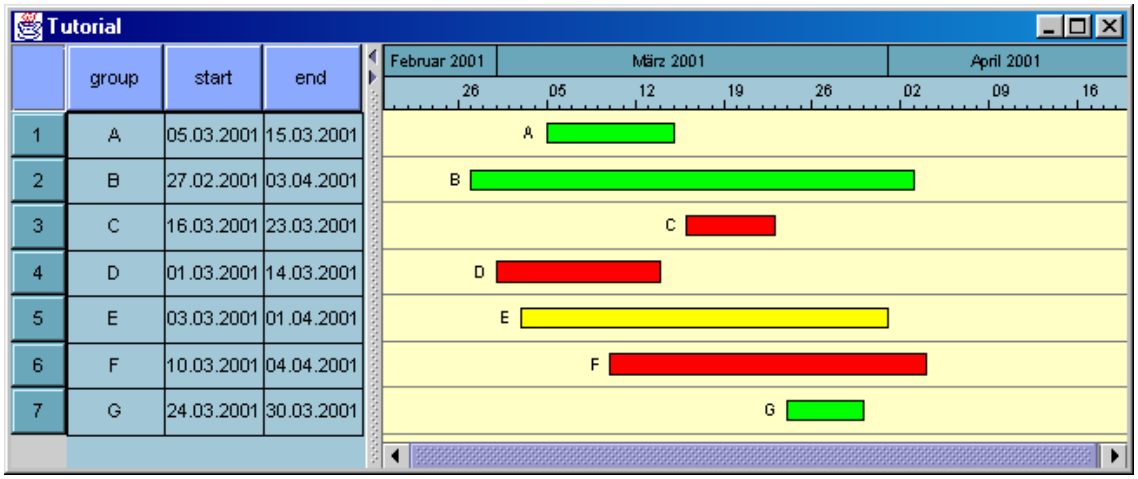
Durch weiteres Einlesen von Dateien bei bereits vorhandenen Daten oder bereits geöffneten Dateien können Sie weitere Zeitspannen zu bereits bestehenden Profilen dazu laden, um z.B. Feiertage zu bereits bestehenden Wochenenden einzulesen. Dies gilt nur, wenn die eingelesene Datei kein Schema enthält (beim Abspeichern musste der saveScheme-Parameter auf **false** gesetzt sein), andernfalls werden alle in der AppData vorhandenen Daten gelöscht, d.h. es erfolgt kein Hinzuladen, sondern ein Überschreiben der im Arbeitsspeicher befindlichen Daten.

Nachdem Sie nun wissen, wie Kalenderdaten abgespeichert werden können, wenden wir uns der Farbgebung im Gantt-Diagramm zu.

5.6 Dynamische Farbgebung

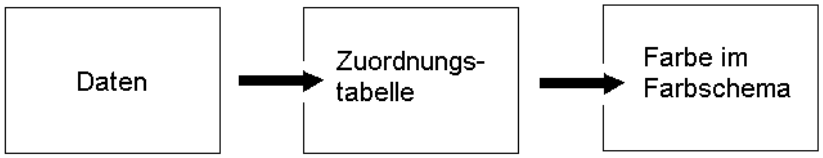
Bisher haben Sie gelernt, dass Sie verschiedenen Elementen des Gantt-Diagramms Farben über Farbschemata zuweisen können: dem Gantt-Graphen, der Zeitskala, der Tabelle und den Knoten. Diese Farbgebungen sind statisch, dies bedeutet z.B. für die Knoten, dass alle unterschiedslos dieselben Farben zeigen. Sowohl bei Knoten als auch bei Gitternetzen im Hintergrund des Gantt-Graphen kann es aber sinnvoll sein, die Farbgebung dynamisch zu gestalten. So könnten besondere Knoten eine bestimmte Farbe

annehmen, oder besondere Gitternetzlinien im Hintergrund des Gantt-Graphen - etwa solche, die Feiertage darstellen – andersfarbig markiert sein.



Im obigen Bild sehen Sie als Beispiel ein Gantt-Diagramm mit verschiedenfarbigen Knoten, die mit dynamischer Farbgebung hinterlegt sind. Die Knoten gehören verschiedenen Gruppen an, die durch einen Buchstaben gekennzeichnet sind. Alle Gruppen sind grundsätzlich grün eingefärbt, mit Ausnahme der Gruppen C...F. Diese sind rot, wiederum mit der Ausnahme, dass E gelb gefärbt ist. Dieses Beispiel mit relativ komplexen Regeln kann über dynamische Farbgebung umgesetzt werden.

Dynamische Farbgebung ist datengesteuert. Bei Knoten werden diese Daten aus einem festzulegenden Attribut geholt, bei Kalendergittern kommen sie aus der Identifikation (Id) einer Zeitspanne. Der Inhalt des Attributs kann dabei als diskreter Wert oder als Bereich interpretiert werden, so dass Sie - am Beispiel der Knoten - die Möglichkeit haben, Knoten sowohl eines Einzelwertes als auch eines Wertebereiches mit besonderen Farben zu belegen. Die dynamische Farbgebung funktioniert im Grundsatz folgendermaßen: Sie legen eine Farbzuordnungstabelle **NeColorMap** an, die Sie mit Schlüssel- und Farbwerten besetzten. Die Schlüsselwerte steuern die Farbgebung und sind sinnvollerweise die Daten, die Sie aus den Entitäts-Attributen einlesen. Die Schlüsselwerte können Sie als diskrete Werte oder als Bereichswerte eintragen. Die Farbwerte der Zuordnungstabelle sind die Farben, die die Objekte später zeigen sollen. Die Farbzuordnungstabelle wird einer Zuordnungsfarbe **NeMappedColor** bekannt gemacht. Anschließend wird die Zuordnungsfarbe in das Farbschema eingetragen, mit dem Sie die Knoten oder ein Profil belegen.



Wir zeigen Ihnen nun zwei Programmierbeispiele, in denen Sie Knoten bzw. Profile dynamisch mit Farben hinterlegen.

5.6.1 Dynamische Farbgebung bei Knoten

Für das Knotenbeispiel soll die Zielsetzung sein, einfache grüne Knoten datengesteuert rot oder gelb erscheinen zu lassen. Dabei sollen die Knoten, die im Attribut „group“ einen Wert zwischen „C“ und „F“ aufweisen, rot dargestellt, wenn der Wert allerdings „E“ ist, sollen sie gelb eingefärbt werden. Die grüne Farbe beim Knotendesign „Rect“ wird durch die shadedColor des Farbschemas festgelegt, so dass der shadedColor die genannte Zuordnungsfarbe zugewiesen wird. Als erstes wird die Zuordnungstabelle angelegt:

```
NeColorMap theMap = new NeColorMap();
```

Im nächsten Schritt wird die Zuordnungstabelle bestückt. Dabei werden zwei Bereichswerte zugewiesen, von denen der zweite lediglich dazu dient, das Ende des ersten zu definieren. Der dritte zugewiesene Wert ist ein diskreter Wert, beachten Sie bitte die entsprechend unterschiedlichen Aufrufe **addRange** und **addEntry**:

```
// Rot zuweisen für Bereich C bis ∞
theMap.addRange ("C", Color.red);

// Vorangegangenen Bereich hinter F begrenzen
theMap.addRange ("G", null);

// Gelb zuweisen für Wert E
theMap.addEntry ("E", Color.yellow);
```

Zuletzt zugewiesene Werte überlagern vorher zugewiesene, außerdem überlagern diskrete Werte Bereichswerte. Daher wird die Farbe Gelb bei allen Knoten mit dem Wert „E“ im Attribut „group“ erscheinen, obwohl E innerhalb des Bereichs C...F liegt. Im nächsten Schritt wird die dynamische Farbe angelegt. In der Parameterliste wird ihr im ersten Parameter die Grundfarbe mitgeteilt, die erscheinen soll, wenn die Schlüsselwerte der Zuordnungstabel-

le nicht zutreffen, im zweiten Parameter wird die Zuordnungstabelle übergeben und im dritten Parameter das Knotenattribut, das die Schlüsselwerte enthält:

```
NeMappedColor theColor =
                        new NeMappedColor(Color.green, theMap,"group");
```

Schließlich wird das Farbschema angelegt und zugewiesen, wobei die Zuordnungsfarbe im dritten Parameter übergeben wird. Damit ist die **shadedColor** dynamisch, während die restlichen Farben des Farbschemas statisch bleiben:

```
JGColorScheme theScheme =
    new JGColorScheme (Color.blue, Color.orange,theColor, Color.pink);
JGantt1.setNodeColorScheme (theScheme);
```

Natürlich ist auch eine Zuweisung dynamischer Farben für die mainColor, die alternateColor und die shadedAlternateColor ebenso möglich, mit den entsprechenden Konsequenzen für die eventuell zugewiesenen Symbole.

5.6.2 Dynamische Farbgebung bei Profilen

Im zweiten Beispiel stellen wir die dynamische Farbgebung bei einem Profil vor, das das zeitliche Muster für ein Kalendergitter im Gantt-Graphen oder der Zeitskala bildet.

Ein Profil ist durch verschiedene Zeitspannen definiert, die durch eine Identifikationsnummer (Id) gekennzeichnet werden. Diese ist benutzerdefiniert und muss nicht eindeutig sein. Im Kapitel über die Bestückung von Profilen hatten wir 15 Zeitspannen mit 10 verschiedenen Ids angelegt. Für die Farbgebung der Zeitspannen werden in den Farbschemata **timeScaleColorScheme** und **ganttColorScheme** für Arbeitszeiten (activity > 0) bei Id=0 die **shadedColor** verwendet, bei Id > 0 die **alternateColor**. Für Nicht-Arbeitszeiten (activity = 0) werden bei Id=0 die **shadedAlternateColor** verwendet, bei Id > 0 ebenfalls die **alternateColor**:

	ID = 0	ID > 0
Arbeitszeit (activity > 0)	shaded Color	alternateColor
Nicht-Arbeitszeit (activity = 0)	shadedAlternateColor	alternateColor

Es bietet sich an, die **alternateColor** durch eine dynamische Farbe zu ersetzen, weil hier alle Ids (außer 0) mit unterschiedlichen Farben besetzt werden können. Dies erfolgt im untenstehenden Programmierbeispiel. Zunächst wird das bereits bestehende Farbschema des Gantt-Graphen übernommen:

```
JGColorScheme theColorScheme=jGantt1.getGanttColorScheme();
```

Anschließend wird eine neue Zuordnungstabelle angelegt:

```
NeColorMap theMap=new NeColorMap();
```

Nun wird die Zuordnungstabelle mit Werten gefüllt:

```
theMap.addEntry(new Integer (1), Color.gray);
theMap.addEntry(new Integer (2), Color.darkGray);
theMap.addEntry(new Integer (3), Color.yellow);
theMap.addEntry(new Integer (4), Color.pink);
theMap.addEntry(new Integer (5), Color.cyan);
theMap.addEntry(new Integer (6), Color.blue);
theMap.addEntry(new Integer (7), Color.green);
theMap.addEntry(new Integer (8), Color.orange);
theMap.addEntry(new Integer (9), Color.darkGreen);
theMap.addEntry(new Integer (10), Color.red);
```

Nun wird die Zuordnungsfarbe angelegt. Dabei wird die **alternateColor** des Farbschemas im ersten Parameter übergeben, um sicherzustellen, dass diese die Zeitspanne einfärbt, falls keiner der Werte in der Zuordnungstabelle zutrifft. Im zweiten Parameter wird die Zuordnungstabelle übergeben, im dritten das Attribut **ID**, gegen dessen Inhalt die Schlüsselwerte der Zuordnungstabelle abgeglichen werden.

```
NeMappedColor theColor =
  new NeMappedColor(theColorScheme.getAlternateColor(), theMap, "ID");
```

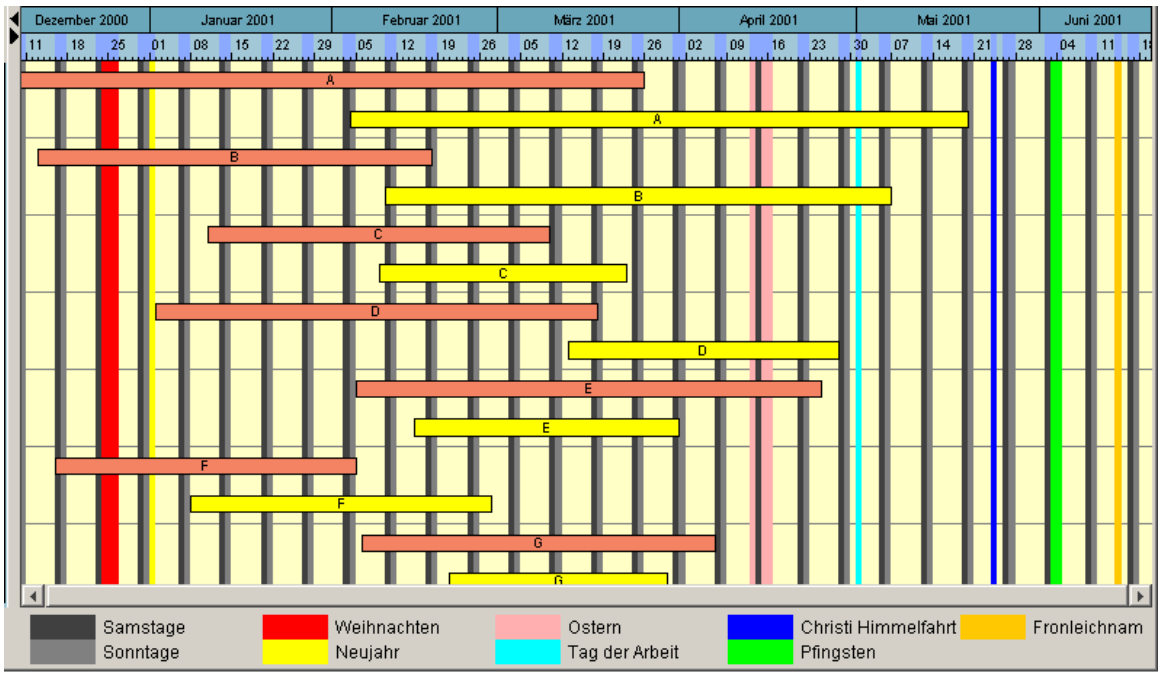
Nun wird im Farbschema des Gantt-Graphen die **alternateColor** durch die Zuordnungsfarbe ersetzt:

```
ColorScheme.setAlternateColor(theColor);
```

Abschließend wird das veränderte Farbschema dem Gantt-Graphen neu zugewiesen:

```
JGantt1.setGanttColorScheme(theColorScheme);
```

Damit ist die dynamische Farbgebung für das Kalendergitter des Gantt-Graphen abgeschlossen:



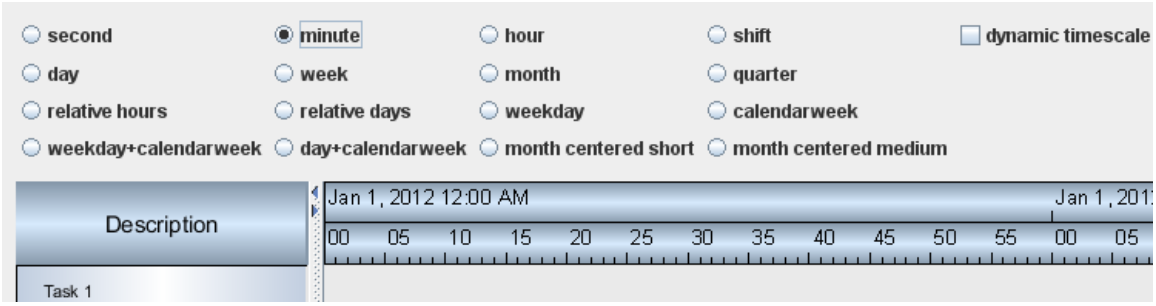
5.7 Erweiterte Optionen der Zeitskala

Wichtige Eigenschaften der Zeitskala sind Typ und Auflösung. Sie ermöglichen es, die Skala an die jeweilige Anwendungssituation optimal anzupassen. Zeitskalentypen, Position, Farbgebung und Beschriftung haben Sie in einem früheren Kapitel schon kennengelernt. Nun lernen Sie besondere Gestaltungsmöglichkeiten der Zeitskala kennen.

5.7.1 Die Formatierung der Zeitskalenbeschriftung

Eine Zeitskala besteht aus horizontalen Bändern, die in der Regel eine von oben nach unten absteigende Skalierung zeigen. Nach der Skalierung des unteren Bandes richtet sich der Name des Zeitskalentyps.

Im mitgelieferten Programmbeispiel **TimescaleTypesTest.jar** können Sie sich einen Überblick über die in VARCHART JGantt vordefinierten Zeitskalentypen verschaffen. Durch Anklicken einer der Auswahloptionen, die Sie im oberen linken Bereich des folgenden Bildes sehen können, schaltet die Zeitskala auf den entsprechenden Typ um.

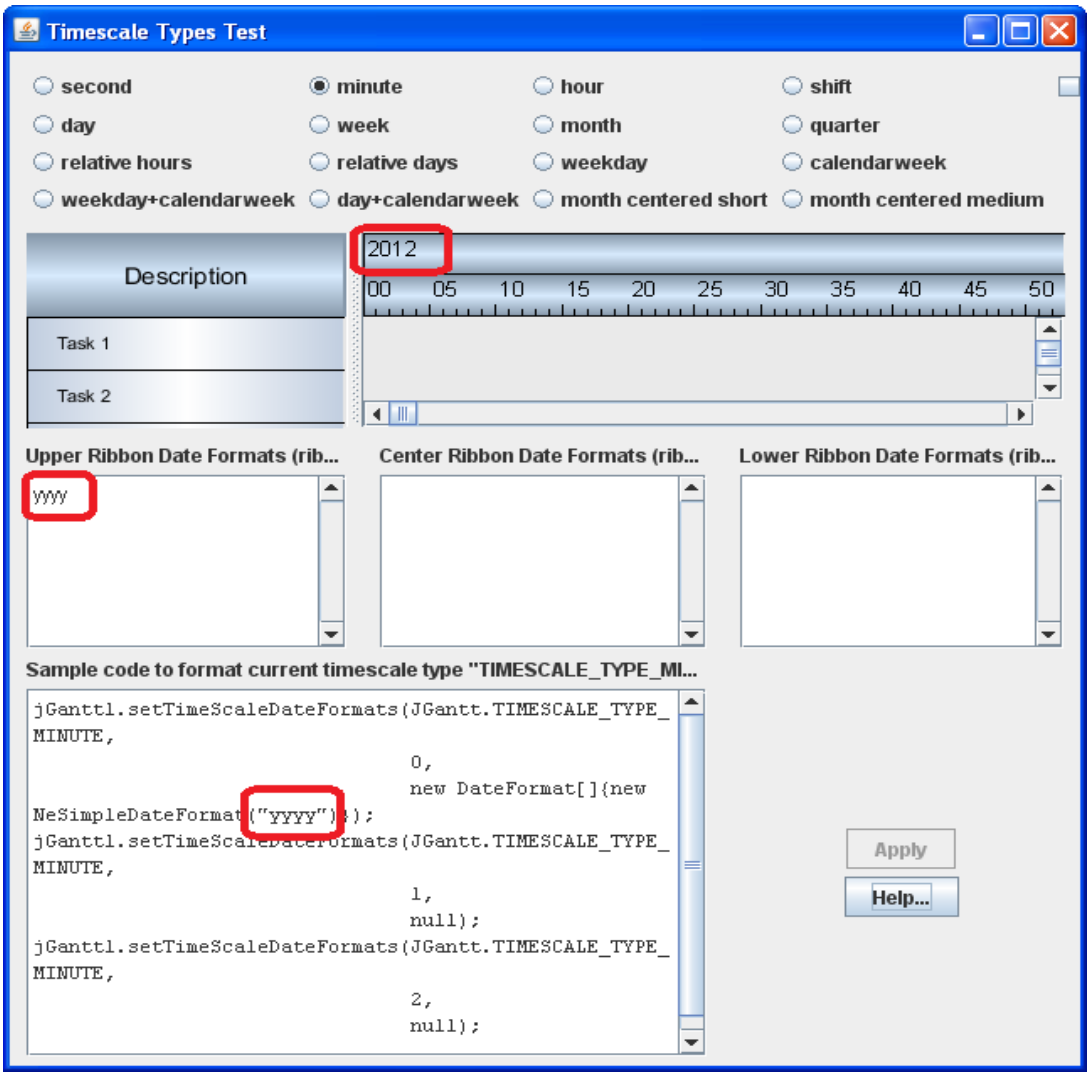


Im Bild oben sehen Sie die Minuten-Skalierung im unteren Band.

Die Beschriftungstexte der Bänder lassen sich für jeden einzelnen Zeitskalentyp formatieren.

Mithilfe des Beispielprogramms können Sie die Formatierung leicht ausprobieren. Dazu finden Sie unterhalb des Gantt-Graphen drei nebeneinander liegende Felder, in die Sie Formatierungsanweisungen eingeben können.

Diese Anweisungen werden in gleicher Weise aufgebaut, wie Sie es aus der Java-Klasse **SimpleDateFormat** kennen.



Nach dem Drücken der Schaltfläche **Apply** werden die Formatierungsanweisungen umgesetzt, und gleichzeitig wird in den darunterliegenden Feldern entsprechender Beispielcode gezeigt, den Sie in eigene Programme übernehmen können.

Das linke Feld bezieht sich auf das oberste Band der Zeitskala, das mittlere Feld auf das darunter liegende Band und das rechte Feld auf das unterste Band – falls vorhanden. Im Bild oben wurde für das obere Band der Minutenskala die Formatierungsanweisung „yyyy“ definiert. Dem entsprechend zeigt das obere Band der Zeitskala im Gantt-Graphen die vierstellige Jahrezahl **2012**.

Bitte beachten Sie, dass das verwendete Datumsformat vom Typ **NeSimpleDateFormat** ist. Diese Unterklasse von **SimpleDateFormat** erlaubt auch die Verwendung von Quartalsziffern, wobei „Q“ für arabische, „R“ für römische Ziffern steht.

Ein Benutzer kann die Zeitskalenauflösung interaktiv mit der Maus verändern. Dazu wird der Mauscursor auf die Skala gesetzt, die linke Maustaste gedrückt und die Maus horizontal nach links oder rechts gezogen. Nach dem Loslassen der Taste zeigt die Skala eine geänderte Auflösung. Ein Ziehen der Maus nach rechts führt zu einer höheren Auflösung, ein Ziehen nach links zu einer niedrigeren.

Nach einer solchen Interaktion kann der gerade verwendete Skalentyp zu einer unübersichtlichen Darstellung führen: Wurde die Skala sehr stark gestaucht, können die Zeiteinheiten und ihre Beschriftungen zu dicht gedrängt stehen; wurde die Skala sehr stark gedehnt, können die Beschriftungen zu weit auseinander liegen und aus dem Sichtbereich geraten.

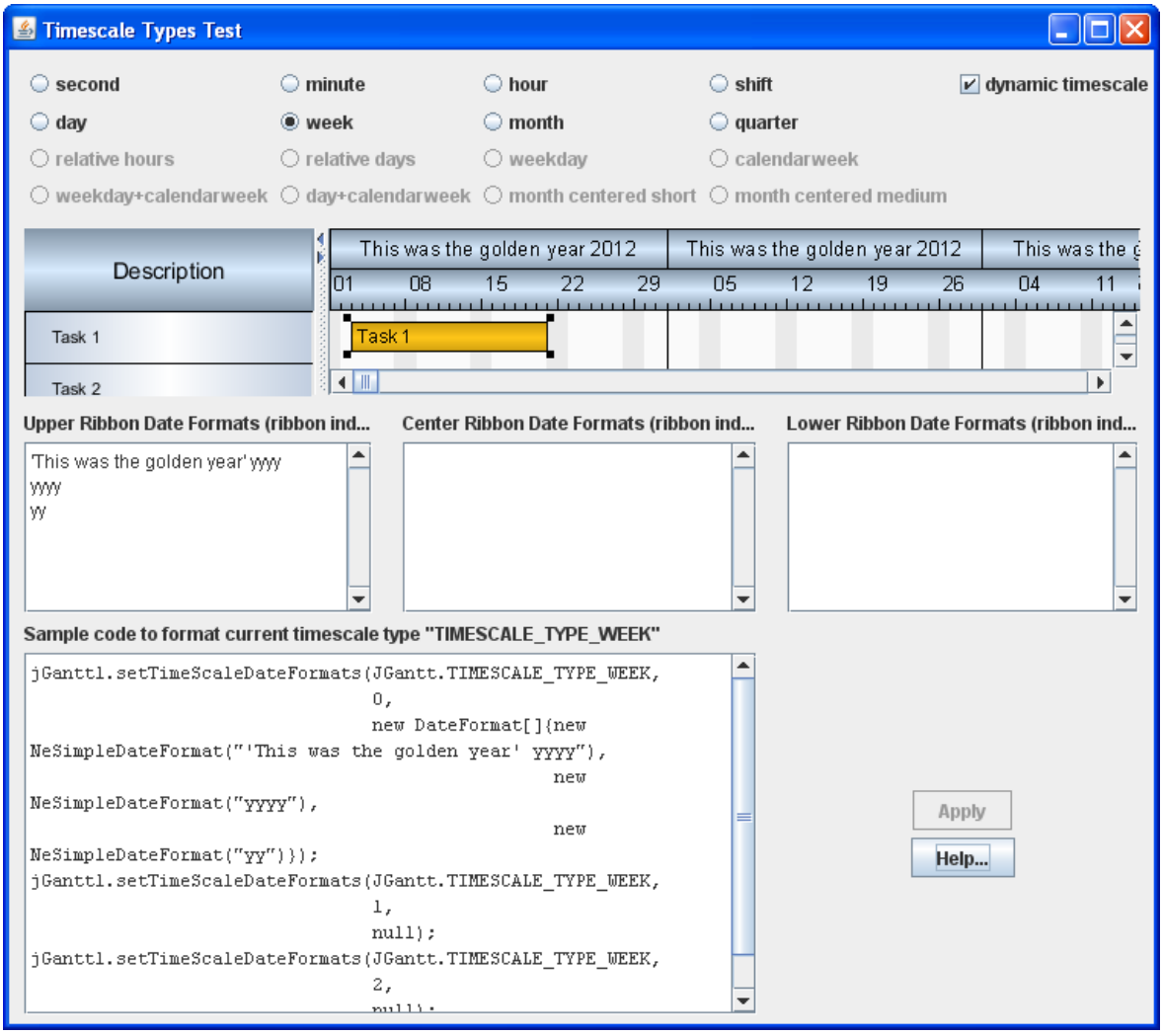
Um diesem Problem zu begegnen, können Sie für ein Band mehrere alternative Formatierungsanweisungen definieren. Die Alternativen geben Sie im Textfeld getrennt durch einen Zeilenumbruch ein. Das Band berücksichtigt in erster Priorität die oberste Zeile. Sinnvollerweise geben Sie hier die Formatierungsanweisung ein, die den längsten Beschriftungstext produziert. Zieht der Benutzer die Auflösung so klein, dass ein Text nicht mehr passt, greift die Zeitskala auf die Formatierungsanweisung der zweiten Zeile zu, usw.

In dem Beispiel unten wurden bei einer Wochenskala für das Jahresband (oberes Band) die drei Alternativen

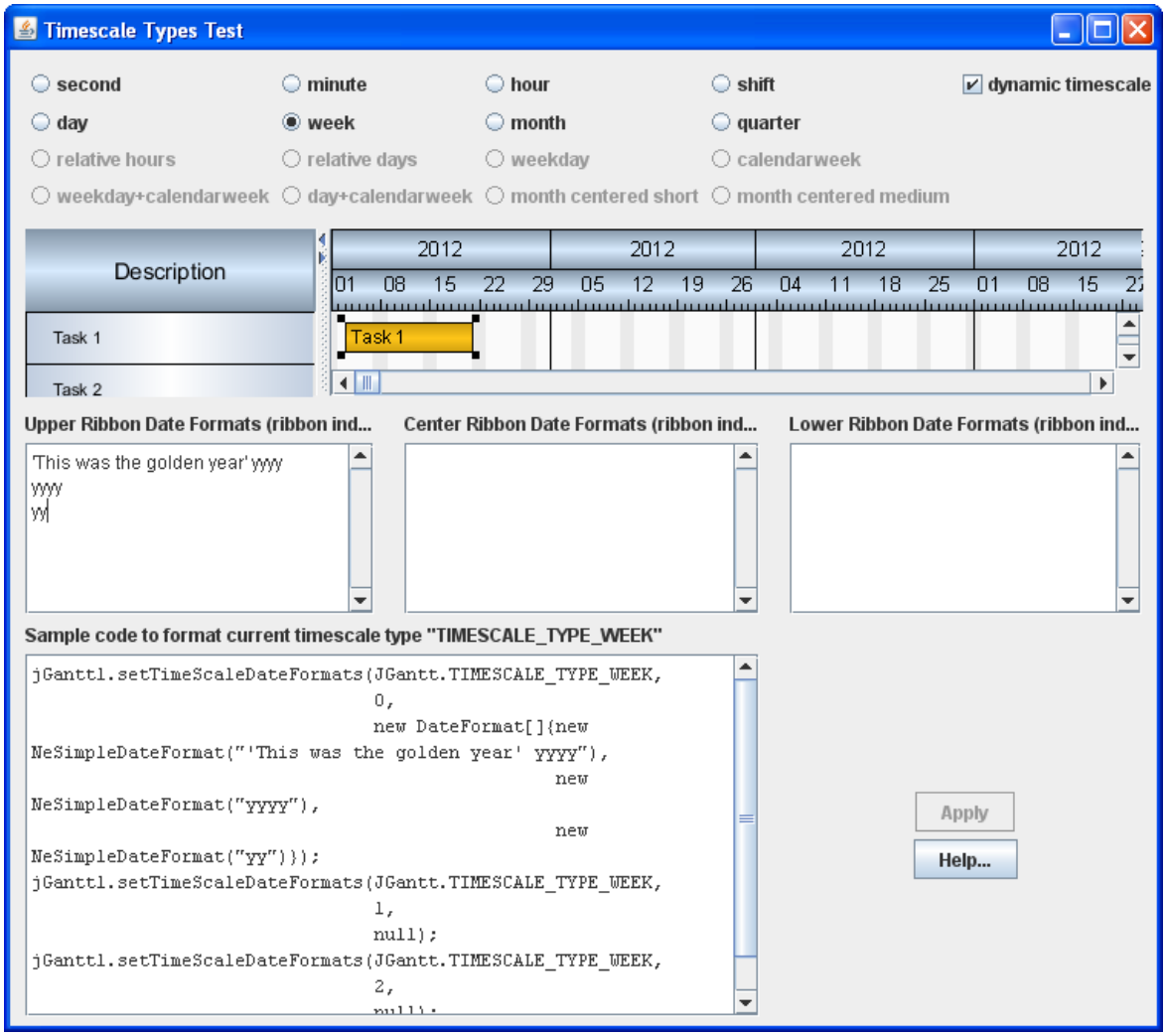
1. ‘This was the golden’ yyyy (bitte beachten Sie die einfachen Anführungsstriche)
2. yyyy
3. yy

eingegeben.

Bei entsprechend hoher Auflösung wird der Text „This was the golden year 2012“ im Jahresband angezeigt.



Verringert der Benutzer die Auflösung, wird im Jahresband nur noch die Jahreszahl „2012“ angezeigt, bei weiterer Reduktion „12“, sofern dies nicht von dem darunterliegenden Band verhindert wird:



5.7.2 Dynamik bei Zeitskalentypen

Nach einer interaktiven Änderung kann der gerade verwendete Skalentyp auch trotz möglicher Formatalternativen unübersichtlich dargestellt sein, wenn die Stauchung oder Dehnung zu groß war. Die nächste Stufe der Problemvermeidung ist der dynamische Skalenwechsel.

Es gibt es beim JGantt-Objekt die boole'sche Eigenschaft **TimeScaleDynamic**. Wenn Sie diese auf **true** setzen, schaltet die Skala automatisch zwischen geeigneten Typen um:

```
jGantt1.setTimeScaleDynamic(true);
```

Im Beispielprogramm setzen Sie die gewünschte Option in einem Kontrollkästchen:

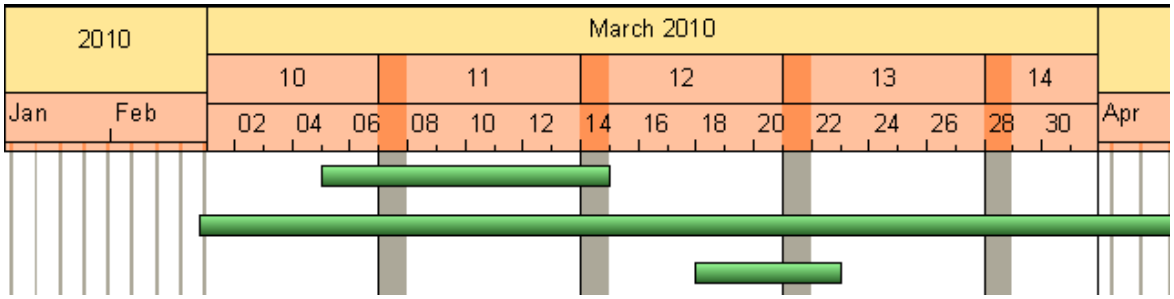
Beachten Sie dabei bitte, dass nur zwischen den Typen **SECOND**, **MINUTE**, **HOUR**, **SHIFT**, **DAY**, **WEEK**, **MONTH** und **QUARTER** gewechselt wird. Im Programmbeispiel sind daher die verbleibenden Typen deaktiviert. Sie können dann bei einer interaktiven Änderung der Skalenauflösung beobachten, wie sich der Typ ggf. ändert.

Die Kombination aus alternativ definierten Formaten und der Zeitskalendynamik bietet Ihnen die Möglichkeit, die Zeitskala bei interaktiven Veränderungen der Auflösung stets aussagekräftig beschriftet zu halten.

5.7.3 Zeitskalenabschnitte

Bei Zeitskalen, die sich über einen größeren Zeitraum erstrecken, besteht häufig die Anforderung, einen bestimmten Ausschnitt des gesamten dargestellten Zeitraums zu vergrößern. Beispielsweise sollen nur die Vorgänge im aktuellen Monat eines Projektes betrachtet werden, das sich über ein ganzes Jahr erstreckt. In diesem Fall können Sie zwar den Zeitskalentyp auf "Tage" oder gar "Stunden" mit geeigneter Auflösung setzen, um den fokussierten Monat detailliert betrachten zu können, doch geht dabei der Überblick über das gesamte Projekt verloren. Wählen Sie hingegen eine gröbere Einteilung - z.B. "Wochen" oder "Monate" -, bleibt zwar die Übersicht über das Projekt erhalten, doch der genaue Einblick in die Planungen des aktuellen Monats ist nicht mehr möglich.

In dieser Situation helfen Zeitskalenabschnitte weiter. Mit ihnen lässt sich eine Skala in beliebig viele Abschnitte unterteilen. Jeder Abschnitt kann einen anderen Typ und eine andere Auflösung zeigen. Damit können Sie im oben skizzierten Beispiel einen Abschnitt vom Typ "Tag" für den aktuellen Monat bilden. Die vor bzw. nach diesem Monat liegenden Zeiträume werden durch zwei weiter Abschnitte vom Typ "Monat" dargestellt. Auf diese Weise können Sie eine "Zeitlupe" schaffen, die einen bestimmten Bereich der Zeitachse visuell vergrößert:



Zeitskalenabschnitte werden über die API angelegt, aufgehoben und konfiguriert. Folgender Befehl unterteilt eine Zeitskala am 1. März 2010 in zwei Abschnitte:

```
int sectionIndex = jGantt1.addTimeScaleSection(new GregorianCalen-
dar(2010, 2, 1).getTime().getTime());
```

Als Rückgabewert liefert addTimeScaleSection den Index des neuen Abschnitts rechts von der Datumsgrenze. Die Abschnitte einer Skala werden, bei 0 angefangen, fortlaufend durchnummeriert. Über die Indizes können Sie jeden Abschnitt einzeln konfigurieren. So lässt sich beispielsweise der Typ des dritten Abschnitts wie folgt festlegen:

```
jGantt1.setTimeScaleSectionTypes(2,
jGantt1.TIMESCALE_TYPE_DAY_CALENDARWEEK);
```

Einen Abschnitt der Zeitskala können Sie auch wieder auflösen. Dabei muss festgelegt werden, wie der Zeitbereich des aufzulösenden Abschnitts künftig dargestellt werden soll. Zwei Möglichkeiten stehen zur Verfügung: Entweder schließt dieser Zeitbereich an den vorhergehenden oder an den nachfolgenden Skalenabschnitt an. Hier die entsprechenden API-Befehle:

```
jGantt1.removeTimeScaleSection(2,
jGantt1.TIMESCALE_SECTION_REPLACE_BY_PREVIOUS);
```

bzw.

```
jGantt1.removeTimeScaleSection(2,
jGantt1.TIMESCALE_SECTION_REPLACE_BY_NEXT);
```

Auch Gantt-Gitter (nicht aber Kalender-Gitter) können Sie abschnittsweise definieren. Analog zu den Möglichkeiten auf der gesamten Zeitskala können Sie für jeden Abschnitt den Typ und die Gittereinstellungen auf "dynamisch" setzen. Das bedeutet, dass JGantt selbstständig für einen Abschnitt den geeigneten Abschnittstyp und den dazu passenden Gittertyp wählt.

Zur Laufzeit kann die Auflösung jedes einzelnen Abschnitts interaktiv mit der Maus verändert werden. Auch eine dynamische Gestaltung aller Abschnitte der Zeitskala ist möglich:

```
jGantt1.setTimeScaleDynamic(true);
```

Die Zeitskalenabschnitte wirken sich natürlich nicht nur auf die Darstellung von oberer/unterer Zeitskala und Gantt-Diagramm aus, sondern auch auf die Visualisierung des Histogramms. Zur weiteren Vertiefung dieser Thematik empfehlen wir, das Beispiel **TimeScaleSections.jar** aus der Beispielsammlung zu betrachten.

Sie haben nun vertiefend besondere Eigenschaften der Zeitskala kennengelernt. Damit können wir uns nun den Verbindungen im Gantt-Graphen zuwenden.

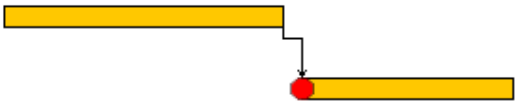
5.8 Verbindungen im Gantt-Graphen

Vorgänge in einem Gantt-Diagramm sind häufig voneinander abhängig. Die Beziehungen zwischen den Vorgängen können verschiedene Qualitäten annehmen:

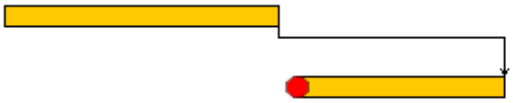
- Ein Vorgang darf erst beginnen, wenn ein anderer Vorgang begonnen hat (Start-Start-Beziehung).
- Ein Vorgang darf erst enden, wenn ein anderer Vorgang begonnen hat (Start-Ende-Beziehung).
- Ein Vorgang darf erst enden, wenn ein anderer Vorgang geendet hat (Ende-Ende-Beziehung).
- Ein Vorgang darf erst beginnen, wenn ein anderer Vorgang geendet hat (Ende-Start-Beziehung).

Für Anwendungsprogramme mit derartigen Abhängigkeiten ist es sinnvoll, die Beziehungen zwischen den Knoten als verbindende Linien zu visualisieren. VARCHART JGantt bietet die Möglichkeit, die vier genannten Qualitäten als entsprechende Verbindungstypen darzustellen:

Start-Start-Verbindung



Start-Ende-Verbindung



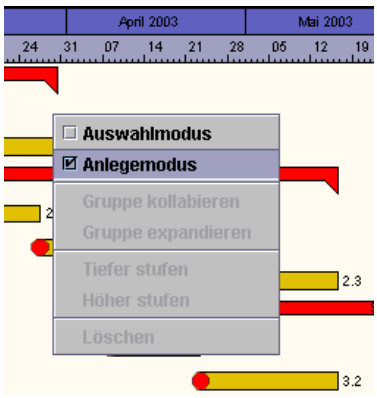
Ende-Start-Verbindung

Ende-Ende-Verbindung

In den folgenden Kapiteln werden Sie erfahren, wie Verbindungen funktionieren und welche Interaktionen seitens des Benutzers mit ihnen möglich sind. Danach werden wir den entsprechenden Quellcode behandeln. Im Lieferumfang des Software-Pakets von VARCHART JGantt finden Sie ein Beispielprogramm mit dem Namen **links.java**, das den hier erläuterten Quellcode enthält.

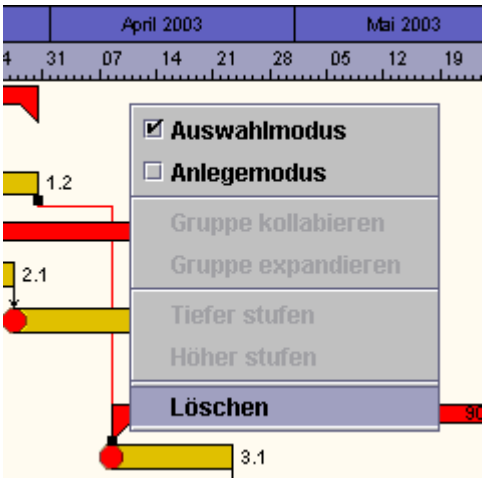
5.8.1 Interaktionen mit Verbindungen

Als Anwender können Sie über das Kontext-Menü (rechte Maustaste) des Gantt-Diagramms zwischen dem Selektieren und dem Anlegen wählen. Dabei gelten diese Modi sowohl für Knoten als auch für Verbindungen, sofern Verbindungen programmiert wurden (darüber erfahren Sie im nächsten Kapitel mehr):

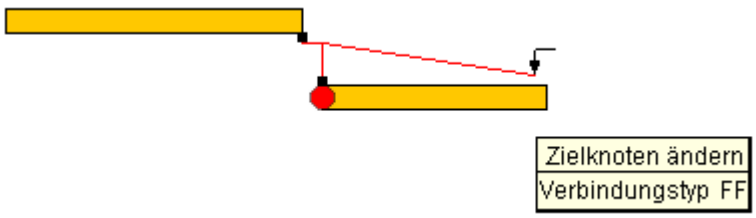


Nach dem Auswählen des Anlege-Modus ist es abhängig von der Maus-Position, ob Knoten oder Verbindungen angelegt werden. Auf den freien Flächen des Gantt-Graphen deutet das Fadenkreuz des Cursors an, dass Knoten angelegt werden können.Sobald Sie die Maus auf einen Knoten verschieben, wandelt sich das Fadenkreuz zu einem Bleistift-Symbol. Es indiziert, dass Sie nun über Drag & Drop eine Verbindung von einem Knoten zu einem anderen zeichnen können. Eine Phantomlinie unterstützt Sie dabei. Sie verdeutlicht,

wo an einem Knoten Sie die Verbindung angesetzt haben und wo Sie sie hinführen. Die Position von Startpunkt oder Endpunkt der Verbindung innerhalb eines Knotens entscheidet über den Verbindungstyp. Positionieren Sie die Maus in der vorderen Hälfte eines Knotens, erzeugen Sie eine Anfangs-Qualität. Positionieren Sie die Maus in der hinteren Hälfte eines Knotens, erzeugen Sie eine Ende-Qualität.



Angelegte Verbindungen können gelöscht werden. Dazu wechseln Sie in den Auswahl-Modus, markieren die Verbindung und löschen sie über die Entfernen-Taste oder über das Kontext-Menü. Sie können eine bestehende Verbindung verschieben und damit ggf. ihren Verbindungstyp ändern:



Dazu klicken Sie mit der Maus auf einen Markierungspunkt und ziehen diesen zum anderen Ende des Knotens oder zu einem anderen Knoten. Im obigen Beispiel wurde der untere Markierungspunkt angeklickt und vom Anfang zum Ende des Knotens verschoben. Damit ändert sich der Verbindungstyp von Ende-Start (engl. Finish-Start) zu Ende-Ende (engl. Finish-Finish).

5.8.2 Programmierung von Verbindungen

Als Vorlage für die Programmierung von Verbindungen steht Ihnen das Beispielprogramm **Links.java** zur Verfügung, das mit der VARCHART JGantt-Software ausgeliefert wird. Das Programmieren von Verbindungen umfasst im Wesentlichen die Anlage und Haltung der Verbindungsdaten. Die Verwaltung der Benutzer-Interaktionen erfolgt von VARCHART JGantt automatisch. Für die Datenverwaltung legen Sie zunächst ein Entitäten-Set an, in dem Sie Attribute definieren, die die Identifikationsnummer des Quell- und Zielknotens enthalten. Ein weiteres Attribut enthält den Verbindungstyp.

```
// Anlegen des Entitäten-Sets "links"
NeIEntitySet links = appData.createEntitySet("links");

// Attribute für Quellknoten, Zielknoten und Verbindungstyp anlegen
links.addEntityAttribute("pre", String.class, "Source node");
links.addEntityAttribute("suc", String.class, "Target node");
links.addEntityAttribute("type", String.class, "Type of link");
```

Jetzt geben Sie dem JGantt bekannt, welches der vorhandenen Entitäten-Sets dasjenige für die Verbindungen ist und definieren die Funktion der angelegten Attribute:

```
// Bekanntgabe von Entitäten-Set und Attributen
jGantt1.setLinkSetName("links");
jGantt1.setLinkSourceNodeBy("pre");
jGantt1.setLinkTargetNodeBy("suc");
jGantt1.setLinkTypeBy("type");
```

Die Interaktionen mit Verbindungen sind nur möglich, wenn die Aufrufe **setLinkSetName**, **setLinkSourceNodeBy** und **setLinkTargetNodeBy** gesetzt sind, d.h. wenn Verbindungen sichtbar sind. Da ein Knoten in der Regel vier Termine besitzt, eine Verbindung hingegen für ihren Anfang und ihr Ende nur zwei Termine benötigt, legen Sie nun fest, welche der vorhandenen Knotentermine als Ansatzpunkte der Verbindung verwendet werden sollen. Dazu geben Sie zwei Indizes an, von denen der erste auf den Ansatztermin des Verbindungsanfangs und der zweite auf den Ansatztermin des Verbindungsendes zeigt. Standardmäßig sind dies die Indizes 0 und 1:

```
jGantt1.setLinkNodeDateIndexes(new int [] {0,1});
```

Anschließend legen Sie eine Vorlage-Entität an und bestücken sie mit Daten:

```
// Vorlage-Entität anlegen
NeIEntity linkTemplate = links.getTemplateEntity();
```

```
// Bestückung der Vorlage-Entität mit Daten
linkTemplate.setValue("pre", "Node 2");
linkTemplate.setValue("suc", "Node 3");
linkTemplate.setValue("type", "FS");
```

Nun wird aus der Vorlage-Entität die endgültige Entität für die erste Verbindung generiert:

```
//Generierung einer Verbindung mit neuer Identifikationsummer
links.createEntity(Frame1.makeLinkID());
```

Für eine weitere Verbindung werden zwei der drei Attribute aus der Vorlage-Entität neu überschrieben und die Verbindung generiert:

```
linkTemplate.setValue("pre", "Node 5");
linkTemplate.setValue("suc", "Node 6");
links.createEntity(Frame1.makeLinkID());
```

Damit haben Sie nun grundlegende Kenntnisse im Programmieren von Verbindungen erworben.

5.9 Das Histogramm

Ein Histogramm dient im Regelfall dazu, Ihrem Benutzer eine grafische Übersicht über die Auslastung seiner Ressourcen aus dem Gantt-Graphen zu geben. Stufen- oder Linienkurven eines Histogramms stellen die addierten Werte bestimmter Aktivitäten im Gantt-Diagramm dar.

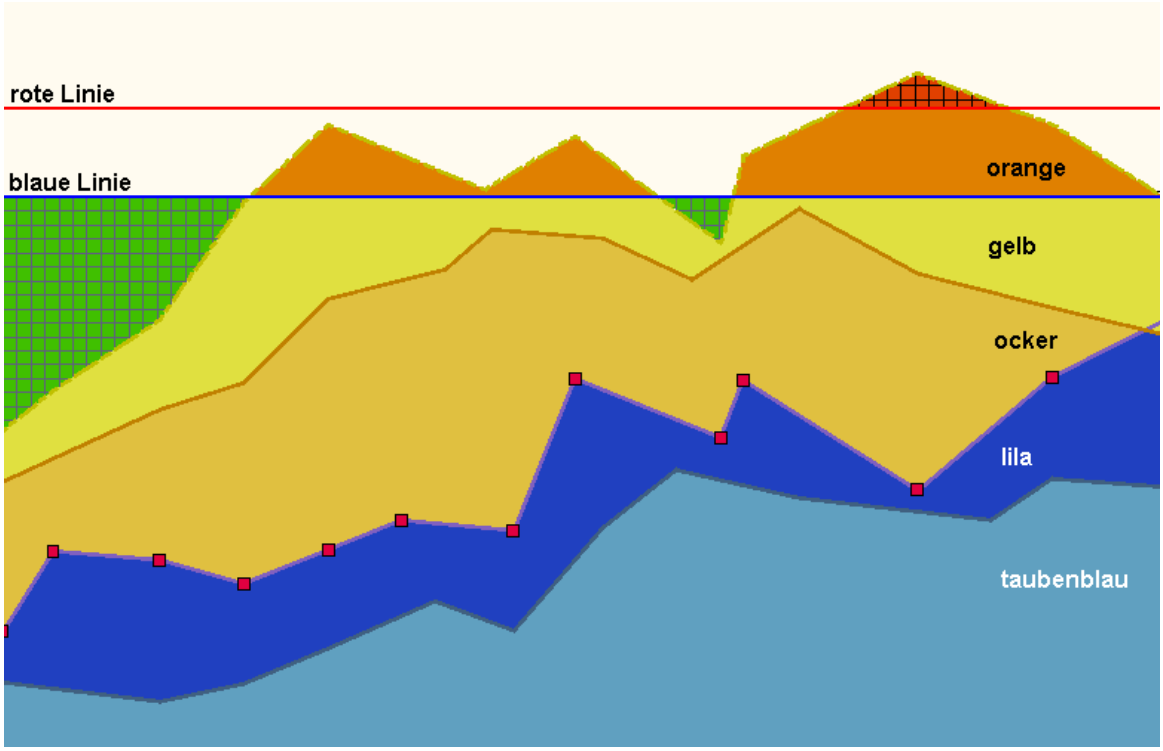
VARCHART JGantt bietet Ihnen die Möglichkeit, ein Histogramm mit mehreren Kurven zu darzustellen, die die jeweilige Auslastung bestimmter Vorgänge anzeigen.

5.9.1 Flächen, Kurven und Grenzwertlinien

Beispielhaft werden wir Ihnen nun die Programmierung des untenstehenden Beispiels zeigen.

Es enthält 4 Linienkurven, die jeweils eine anders gestaltete Fläche mit einer anderen Kurve bilden. Auf der zweiten Kurvenlinie von unten sind die Punkte mit einem roten Rechteck-Symbol hervorgehoben. Die oberste Kurve ist oberhalb der blauen (unteren) Grenzwertlinie orange eingefärbt und oberhalb der (oberen) roten Linie rot mit schwarzer Kreuzschraffierung. Zudem zeigt

sie unterhalb der blauen Linie eine gelbe Fläche vor grünem Hintergrund mit grauer Kreuzschraffierung. Nach unten hin folgen weitere 3 Kurven mit ockerfarbener, lila und mit taubenblauer Fläche. Ihre Linien sind von unterschiedlicher Qualität und Farbe.



Histogramme können Sie über eine eigene Komponente programmieren, in der 10 Klassen und Interfaces zusammengefasst sind. Für unser Histogramm beginnen Sie damit, die 6 Kurven mit Daten zu bestücken. In einer realen Anwendung lesen Sie diese aus einer Datenbank oder einem Entitäten-Set ein; wir begnügen uns damit, die Arrays mit erzeugten Daten zu bestücken. Dabei listen wir beispielhaft den Code für eine Kurve mit 12 Punkten und eine Grenzwertlinie mit 2 Punkten auf:

```
private NeArrayCurveData createSampleCurveData (int index)
{
//Bestückung der Kurve mit dem Index 0
NeArrayCurveData acd = new NeArrayCurveData();

if (index == 0)
   {
   acd = new NeArrayCurveData ();
   acd.setXValue(0, new GregorianCalendar(2004, 11, 20).getTime().
   getTime());

   acd.setYValue(0, 6.0f);
```

```
   acd.setXValue(1, new GregorianCalendar(2005, 0, 17).getTime().
   getTime());

   acd.setYValue(1, 4.3f);
   acd.setXValue(2, new GregorianCalendar(2005, 1, 1).getTime().
   getTime());

   acd.setYValue(2, 5.9f);
   acd.setXValue(3, new GregorianCalendar(2005, 1, 16).getTime().
   getTime());

   acd.setYValue(3, 9.0f);
   acd.setXValue(4, new GregorianCalendar(2005, 2, 7).getTime().
   getTime());

   acd.setYValue(4, 13.3f);
   acd.setXValue(5, new GregorianCalendar(2005, 2, 21).getTime().
   getTime());

   acd.setYValue(5, 10.6f);
   acd.setXValue(6, new GregorianCalendar(2005, 3, 6).getTime().
   getTime());

   acd.setYValue(6, 19.9f);
   acd.setXValue(7, new GregorianCalendar(2005, 3, 19).getTime().
   getTime());

   acd.setYValue(7, 25.0f);
   acd.setXValue(8, new GregorianCalendar(2005, 4, 11).getTime().
   getTime());

   acd.setYValue(8, 22.5f);
   acd.setXValue(9, new GregorianCalendar(2005, 5,14).getTime().
   getTime());

   acd.setYValue(9, 20.5f);
   acd.setXValue(10, new GregorianCalendar(2005, 5,25).getTime().
   getTime());

   acd.setYValue(10, 24.3f);
   acd.setXValue(11, new GregorianCalendar(2005, 6,17).getTime().
   getTime());

   acd.setYValue(11, 23.4f);
   }

// Bestückung Grenzwertlinie mit Index 1
else if (index==1)
   {
   acd.setXValue(0, new GregorianCalendar(2005, 5,25).getTime().
   getTime());

   acd.setYValue(0, 33.3f);
   acd.setXValue(1, new GregorianCalendar(2005, 6,17).getTime().
   getTime());

   acd.setYValue(1, 33.3f);
   }

else if (index==2)
```

```
   { ……

    // Wir verzichten auf die Darstellung
    // des Codes der restlichen Kurven

   }
return acd;}
```

Bevor Sie die Kurvendaten verarbeiten, legen Sie die Größe und Hintergrundfarbe des Symbols aus der zweiten Kurve (von unten) fest:

```
public void curveExample ()
{
//Symbol Nr. 10 ist ein Viereck
NeSymbol sym1 = new NeSymbol(10);

//Größe u. Hintergrundfarbe des Symbols
sym1.setExtension(new Dimension (300, 300));
sym1.setAreaStyle(new Color (1.0f, 0.0f, 0.3f));
```

Dann legen Sie für alle Kurven die Defaultwerte für Farbe, Liniendicke und Linientyp fest:

```
// Default-Farbe violett, Dicke 3/100 mm, kurz gestrichelte Linie
NeCurveStyle curveStyle = new NeCurveStyle(new NeLineStyle
 (new Color (1.0f, 0.0f, 0.5f), 3, NeIDrawinCostants.NE_LINE_TYPE_1));
```

Nun legen Sie aus den vorgegebenen Daten insgesamt 6 Linienkurven (zwei davon sind Grenzwertlinien) in einem Array an und gestalten ihr Aussehen. Hier legen Sie die unterste Kurve (Kurve 0) an.

```
// Unterste Kurve (Kurve 0)
NeArrayCurveData arrayCurveCD0 = createSampleCurveData (0);
myLineCurves[0] = new NeLineCurve();
myLineCurves[0].setCurveData (arrayCurveCD0);

// Die Kurve wird von der X-Achse an ...
 myLineCurves[0].setCurveStyle
    (0, new NeCurveStyle(NeCurveStyle.APPLY_IF_GREATER_THAN,null,

// ... taubenblau gefüllt ...
     new Color (0.4f, 0.6f, 0.8f),

// ... und erhält eine graublaue, 7/10 mm breite, durchgezogene Linie
     new NeLineStyle (new Color (0.2f, 0.4f, 0.6f), 70,
NeIDrawingConstants.NE_LINE_TYPE_0)));

// Kurve wird ins Histogramm gezeichnet
neHistogram1.addCurve(myLineCurves[0]);
```

Hier legen Sie Kurve 1 an:

```
// Kurve 1
```

```
NeArrayCurveData arrayCurveCD1 = createSampleCurveData (6);
myLineCurves[1] = new NeLineCurve();
myLineCurves[1].setCurveData(arrayCurveCD1);

// Die Kurve wird von Kurve 0 an...
myLineCurves[1].setCurveStyle(0, new NeCurveStyle
(NeCurveStyle.APPLY_IF_GREATER_THAN, myLineCurves[0],

//... lila ausgefüllt ...
      new Color (0.15f, 0.25f, 0.8f),
//... mit dunkellila Linienfarbe
      new NeLineStyle (new Color (0.5f, 0.4f, 0.7f), 70,

NeIDrawingConstants.NE_LINE_TYPE_0)));

// Kurve wird ins Histogramm gezeichnet
neHistogram1.addCurve(myLineCurves[1]);
```

Hier legen Sie Kurve 2 an:

```
//Kurve 2
rrayCurveCD1 = createSampleCurveData (4);
myLineCurves[2] = new NeLineCurve();
myLineCurves[2].setCurveData(arrayCurveCD1);

//Die Kurve wird von Kurve 1 an...
      myLineCurves[2].setCurveStyle (0, new NeCurveStyle
(NeCurveStyle.APPLY_IF_GREATER_THAN, myLineCurves[1],

//... sandfarben ausgefüllt ...
new Color (0.9f, 0.7f, 0.2f),

//... mit ockerfarbener Linienfarbe
      new NeLineStyle (new Color (0.7f, 0.53f, 0.03f), 70,
NeIDrawingConstants.NE_LINE_TYPE_0)));

// Kurve wird ins Histogramm gezeichnet
neHistogram1.addCurve(myLineCurves[2]);
```

Hier legen Sie die blaue Grenzwertlinie an:

```
// Kurve 4
myLineCurves[4] = new NeLineCurve();
myLineCurves[4].setCurveData(createSampleCurveData (3));

// Blauer Strich ohne Referenz zu einer anderen Kurve
NeICurveStyle curSty = new NeCurveStyle(new NeLineStyle
                      (new  Color (0.0f, 0.0f, 1.0f), 50,
                    NeIDrawingConstants.NE_LINE_TYPE_0));
myLineCurves[4].setCurveStyle (0, curSty);

// Kurve 4 wird erst später gezeichnet, da sie eine
// hohe Zeichenpriorität haben soll
```

Hier legen Sie die rote Grenzwertlinie an:

```
// Kurve 5
myLineCurves[5] = new NeLineCurve();

// Es wird Kurve 4 genommen und 8 zum Y-Wert hinzuaddiert
myLineCurves[5].setCurveData(new NeCalculatedCurveData
(createSampleCurveData (3), 1.f, 8.0f));

// Roter Strich ohne Referenz zu einer anderen Kurve
curSty = new NeCurveStyle(new NeLineStyle
                         (new Color (1.0f, 0.0f, 0.0f), 50,
                          NeIDrawingConstants.NE_LINE_TYPE_0));
myLineCurves[5].setCurveStyle (0, curSty);
```

Hier legen Sie die oberste Kurve an:

```
// Kurve 3 = oberste Kurve
NeArrayCurveData arrayCurveCD2 = createSampleCurveData (5);
myLineCurves[3] = new NeLineCurve();
myLineCurves[3].setCurveData(arrayCurveCD2);

// Die Kurve wird von Kurve 2 an...
myLineCurves[3].setCurveStyle (0, new
NeCurveStyle(NeCurveStyle.APPLY_IF_GREATER_THAN, myLineCurves[2],

// ... hellgelb ausgefüllt ...
new Color (0.85f, 0.85f, 0.2f),

// ... mit senffarbener, unterbrochener Linie
new NeLineStyle (new Color (0.7f, 0.7f, 0.1f), 70,

NeIDrawingConstants.NE_LINE_TYPE_13)));

// Zu der blauen Linie hin ...
myLineCurves[3].setCurveStyle (1, new
     NeCurveStyle(NeCurveStyle.APPLY_IF_GREATER_THAN, myLineCurves[4],

// wird eine orange Fläche gebildet
      new Color (0.9f, 0.5f, 0.0f),

//... mit senffarbener, durchgezogener Linie
      new NeLineStyle (new Color (0.0f, 0.7f, 0.1f), 30,

NeIDrawingConstants.NE_LINE_TYPE_0)));

// Zu der roten Linie hin...
myLineCurves[3].setCurveStyle(2, new
NeCurveStyle(NeCurveStyle.APPLY_IF_GREATER_THAN, myLineCurves[5],

// wird eine dunkelrote Fläche gebildet mit schwarzer Kreuzschraffur
      new NeAreaStyle (new Color (0.0f, 0.0f, 0.0f),

NeDEPainter.FILL_PATTERN_CROSS_WIDE,
new Color (0.9f, 0.2f, 0.0f)), null));

// Unterhalb der blauen Linie...
myLineCurves[3].setCurveStyle(3, new
      NeCurveStyle(NeCurveStyle.APPLY_IF_LESS_THAN, myLineCurves[4],

// wird der Hintergrund grün mit grauer Kreuzschraffur gesetzt
```

```
      new NeAreaStyle (new Color (0.4f, 0.4f, 0.4f),

NeDEPainter.FILL_PATTERN_CROSS_WIDE,
new Color (0.3f, 0.8f, 0.07f)), null));
```

Jetzt werden die Kurve 3 sowie die rote und blaue Grenzwertlinie gezeichnet. Die letztgezeichneten Kurven besitzen die höchste Zeichnungspriorität:

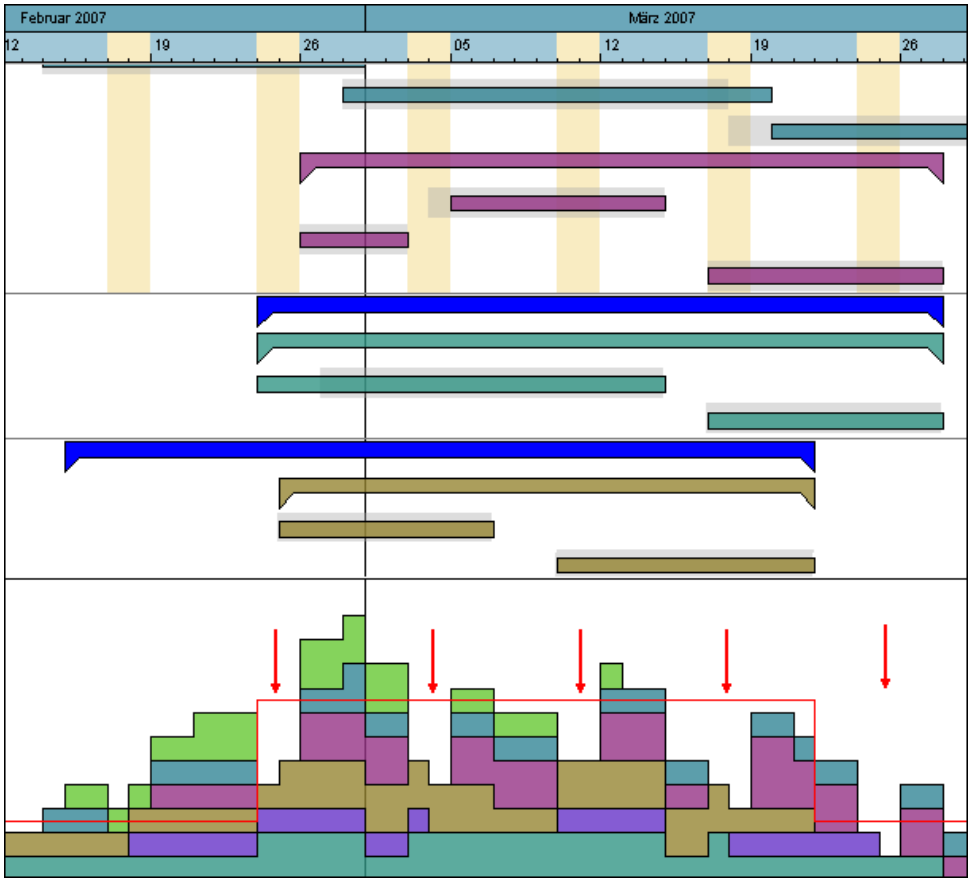
```
// Kurven werden gezeichnet
neHistogram1.addCurve(myLineCurves[3]);
neHistogram1.addCurve (myLineCurves[4]);
neHistogram1.addCurve (myLineCurves[5]);
}
```

Damit sind die Kurven im Histogramm gezeichnet. Jetzt können Sie noch eine kleine Funktion zum An- und Abschalten der Symbole in Kurve 1 schreiben (und diese Funktion ggf. einer Schaltfläche zuweisen):

```
public void toggleSymbols (ActionEvent e)
{
// Schaltet Symbole an oder aus
if (!mySymbolsOn)
   myLineCurves[1].getCurveStyle(0).setLabel(sym1);
else
   myLineCurves[1].getCurveStyle(0).setLabel(null);

mySymbolsOn = !mySymbolsOn;
repaint();
}
```

5.9.2 Kalender im Histogramm

Um ein Histogramm hinsichtlich der arbeitsfreien Zeiten homogen mit dem darüber liegenden Gantt-Graphen erscheinen zu lassen, können Sie auch einem Histogramm denselben Kalender zuweisen, der in Ihrem Gantt-Graphen Arbeitszeiten und arbeitsfreie Zeiten darstellt. Auf diese Weise wird dasselbe Zeitmuster im Gantt-Graphen und im Histogramm abgebildet. Die arbeitsfreien Zeiten, die im Bild unten nur im oberen Drittel des Gantt-Graphen dargestellt sind, bilden sich dementsprechend im Histogramm auch nur partiell ab (im Bild durch die roten Pfeile verdeutlicht):



Folgende Programmierschritte nehmen Sie vor: Zunächst weisen Sie den Kurvendaten den Kalender des JGantt-Objektes zu.

```
CurveData.setCalendar (theCalendar);
```

Dann weisen Sie dem Kalender das bestückte Profil zu. Sie geben es über den Namen eines Attributs bekannt, das das Profil enthält. Das Attribut trägt im Beispiel den Namen "theProfileAttr":

```
setCalendarProfileBy ("theProfileAttr");
```

Für den Fall, dass einem Knoten kein Profil zugewiesen wurde, kommt das Standardprofil zur Anwendung, das Sie hier setzen:

```
SetDefaultCalendarProfile ("theDefaultProfile");
```

Auf diese Weise können Sie arbeitsfreie Zeiten auch im Histogramm darstellen.

6 Interaktionen des Benutzers

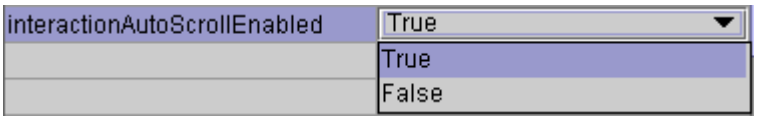
Nachdem Sie sich nun eine Grundlage zum Modellieren eines Gantt Diagramms erarbeitet haben, werden wir Ihnen in diesem Kapitel zeigen, welche Möglichkeiten VARCHART JGantt im Umgang mit Benutzerinteraktionen bietet. Es gibt verschiedene Kategorien von Benutzerinteraktionen, die wir Ihnen in den folgenden Unterkapiteln vorstellen werden.

6.1 Konfiguration von Interaktionen

Das Verhalten von VARCHART JGantt bei Benutzerinteraktionen können Sie über eine Reihe von Eigenschaften beeinflussen, die in der Eigenschaftentabelle aufgeführt sind und mit dem Namensteil **Interaction** beginnen:

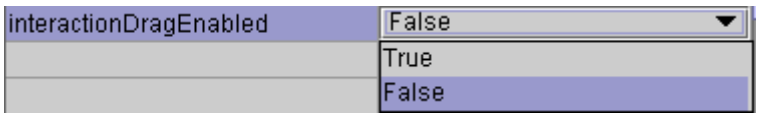
interactionAutoScrollEnabled	True
interactionDragEnabled	False
interactionDropEnabled	True
interactionGroupNodeMoveMode	move nodes or bars
interactionGroupNodeResizeMode	left and right
interactionInfoWindowEnabled	True
interactionLinksAtLeafNodesOnly	True
interactionMode	select nodes
interactionNodeMoveMode	move nodes or bars
interactionNodeResizeMode	left and right
interactionPopupEnabled	True
interactionTimeQuantum	day

Mit der Autoscroll-Funktion können Sie den automatischen Bildlauf einschalten, sobald der Benutzer bei der Interaktion mit einem Diagrammelement den Rand des Diagramms erreicht, z.B. beim Verschieben eines Knotens:




```
// Autoscroll Funktion einschalten
jGantt1.setInteractionAutoScrollEnabled(true);
```

Die beiden folgenden Eigenschaften steuern das Verhalten des JGantt-Objektes während einer drag&drop-Aktion. Mit **interactionDragEnabled** können Sie festlegen, ob das JGantt-Objekt als drag-Quelle fungieren soll:

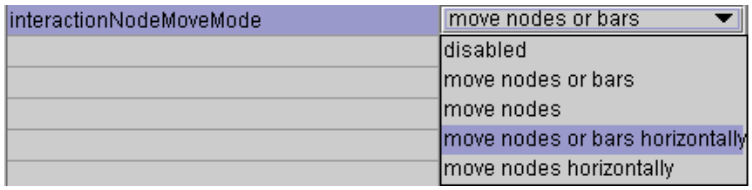


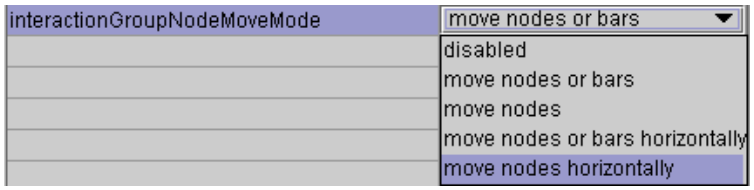

```
//Gantt Objekt darf nicht als Drag-Quelle fungieren
jGantt1.setInteractionDragEnabled(false);
```

Mit der Eigenschaft interactionDropEnabled können Sie festlegen, ob das JGantt-Objekt als Drop-Ziel fungieren soll:

```
//Gantt-Objekt soll nicht als Drop-Ziel fungieren
jGantt1.setInteractionDropEnabled(false);
```

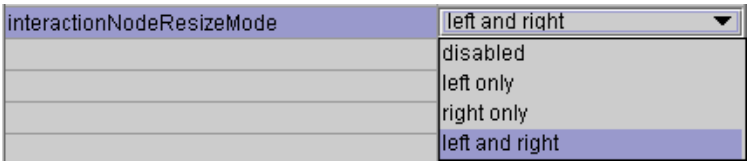
Über den Bewegungsmodus können Sie die volle oder teilweise Beweglichkeit von einfachen Knoten und Gruppenknoten bestimmen:

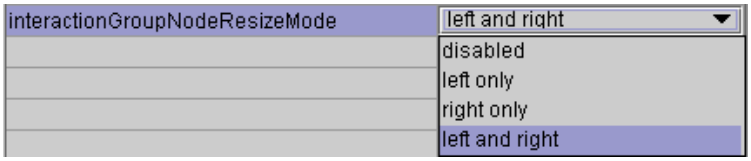





```
//Knoten und Knotenteile dürfen nur horizontal bewegt werden
 jGantt1.setInteractionNodeMoveMode
de.netronic.jgantt.JGantt.INTERACTION_MOVE_MODE_NODES_BARS_HORIZONTALLY);

//Nur ganze Gruppenknoten können horizontal bewegt werden
jGantt1.setInteractionGroupNodeMoveMode
(de.netronic.jgantt.JGantt.INTERACTION_MOVE_MODE_NODES_HORIZONTALLY);
```

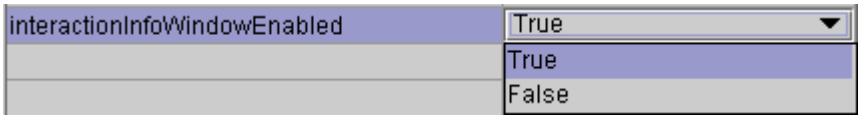
Auch die Option, die Größe von Knoten oder Gruppenknoten zu verändern und falls ja, an welchem Ende, können Sie hier festlegen:





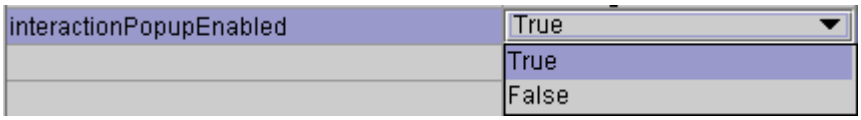

```
// Einfache Knoten können am vorderen und hinteren Ende
// verändert werden
jGantt1.setInteractionNodeResizeMode
(de.netronic.jgantt.JGantt.INTERACTION_RESIZE_MODE_LEFT_AND_RIGHT);

// Gruppenknoten können am vorderen und hinteren Ende verändert werden
jGantt1.setInteractionGroupNodeResizeMode
(de.netronic.jgantt.JGantt.INTERACTION_RESIZE_MODE_LEFT_AND_RIGHT);
```

Wenn der Benutzer mit dem Diagramm interagiert, z.B. beim Verschieben eines Knotens, können Sie das Informationsfenster, das dabei automatisch erscheint, um z.B. die aktuelle Position anzuzeigen, unterdrücken:



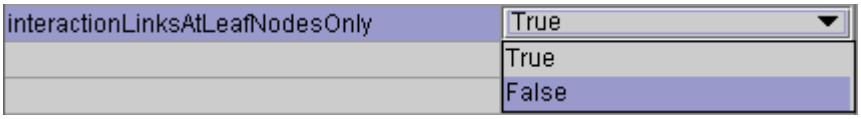

```
// Deaktivieren des Informationsfensters
jGantt1.setInteractionInfoWindowEnabled(false);
```

Dasselbe können Sie mit dem Kontextmenü tun, das nach dem Drücken der rechten Maustaste erscheint:



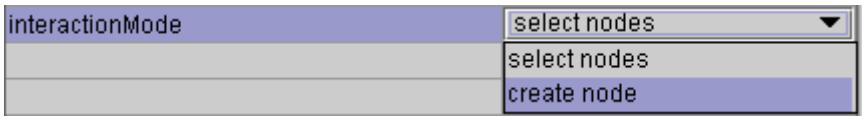

```
// Deaktivierung des Kontextmenüs
jGantt1.setInteractionPopupEnabled(false);
```

Eine weitere Einstellung, die Sie vornehmen können, ist, ob ausschließlich die Blattknoten einer hierarchischen Struktur als Quelle oder Ziel einer Verbindung dienen sollen. In den meisten Applikationen mit hierarchischen Strukturen ist dies sinnvoll und Sie sollten den Default-Wert auf True belassen.



```
// Verbindungen auch an Nicht-Blattknoten zulassen
jGantt1.setInteractionLinksAtLeafNodesOnly(false);
```

Mit der Eigenschaft "InteractionMode" legen Sie fest, ob sich beim ersten Aufruf das JGantt-Objekt im Modus "Knoten anlegen" oder "Knoten selektieren/verschieben" befinden soll. Zur Laufzeit kann der Modus beliebig über das Menü der rechten Maustaste verändert werden.




```
//Dem JGantt-Objekt den Anlege-Modus für Knoten zuweisen
jGantt1.setInteractionMode(JGantt.INTERACTION_CREATE_NODE);
```

Mit der letzten Interaktionseigenschaft legen Sie die Zeitmenge fest, die die Sprunggröße bei der waagerechten Verschiebung eines Objektes bestimmt. Der Wert wird in Millisekunden angegeben, für die gebräuchlichsten Sprungweiten stehend entsprechende Konstanten zur Verfügung:

```
// Sprunginterval über Konstante setzen
jGantt1.setInteractionTimeQuantum(JGantt.INTERACTION_TIMEQUANTUM_WEEK);
// Sprunginterval über die entsprechende Anzahl Sekunden setzen
jGantt1.setInteractionTimeQuantum(604800000);
```

6.2 Steuerung von Interaktionen über Ereignisse

VARCHART JGantt bietet eine Reihe von Ereignissen an, die Sie zur Steuerung von Benutzerinteraktionen verwenden können. Das Steuerungskonzept ist vierstufig. Auf jeder Stufe werden Ihnen eigene Ereignisse zur Verfügung gestellt, die Ihnen weitest gehende Kontrolle über die Benutzeraktionen ermöglichen, die das Anlegen, Verändern und Löschen von Objekten im Diagramm betreffen.

1. Die erste Stufe erfasst beabsichtigte Handlungen des Benutzers. Dieser deutet seine Handlungsabsichten durch das Überfahren eines Gebietes mit der Maus an. Auf dieser Stufe können Sie z.B. bereits eine ursprünglich vorgesehene Cursoränderung unterdrücken, mit der Folge, dass auch die daran gebundenen Interaktionen unterbleiben, z.B. die Anlage eines Knotens in einem Bereich des Diagramms, in dem dies nicht erlaubt ist.
2. Die zweite Stufe umfasst das Herunterdrücken der Maus und das Ziehen eines Objektes mit der Maus. Auch hier können Sie eingreifen und z.B. das Ziehen in eine bestimmte Richtung unterdrücken.
3. Die dritte Stufe umfasst das Loslassen der Maus. Hier können Sie, z.B. wenn eine das Verschieben in einen verbotenen Bereich erfolgte, die Interaktion rückgängig machen oder anderweitig reagieren.
4. Es gibt noch eine vierte Stufe, die allerdings keine direkten Eingriffsmöglichkeiten mehr bietet. Nach Abschluss einer Operation, können Sie, z.B. wenn diese fehlerhaft war, Folgeschritte wie Berechnungen oder Datenbankzugriffe unterdrücken.

Für die Stufe eins, d.h. für beabsichtigte Handlungen, stehen Ihnen die Methoden **prepareCreateInteraction** und **prepareModifyInteraction** zur Verfügung. Die erste bezieht sich auf die Anlage eines Knotens, die zweite auf die Änderung eines vorhandenen Knotens.

Für die Stufe zwei, das Herunterdrücken der Maus und das Ziehen eines Objektes, bei dem sich das Phantom verändert, verwenden Sie die Methode **onPhantomModify**.

Für die Stufe drei, das Loslassen der Maus, stehen Ihnen die Methoden **onObjectModify** und **onObjectCreate** zur Verfügung.

Auf die Stufe vier, nach erfolgter Aktion, können Sie mit den Methoden **objectModified** und **objectCreated** eingreifen und vorgesehene Abläufe verhindern.

Wir werden Sie nun über eine Serie von Programmierbeispielen durch diese vier Stufen führen. Vorab wird ein Listener für die Ereignisse eingerichtet:

```
jGantt1.addNodeChangeListener
(new de.netronic.common.event.NeObjectChangeAdapter()
{
public void prepareModifyInteraction(NeObjectChangeEvent e) throws
                                                    NeVetoException
  {
  jGantt1_prepareModifyInteraction(e);
  }
```

```
public void prepareCreateInteraction(NeObjectChangeEvent e)
throws NeVetoException
  {
  jGantt1_prepareCreateInteraction(e);
  }

public void onPhantomModify(NeObjectChangeEvent e) throws
                                                    NeVetoException
  {
  jGantt1_onPhantomModify(e);
  }

public void onObjectModify(NeObjectChangeEvent e) throws
                                                   NeVetoException
  {
  jGantt1_onObjectModify(e);
  }

public void onObjectCreate(NeObjectChangeEvent e) throws
                                                   NeVetoException
  {
  jGantt1_onObjectCreate(e);
  }

public void objectModified(NeObjectChangeEvent e)
  {
  jGantt1_objectModified(e);
  }

public void objectCreated(NeObjectChangeEvent e)
  {
  jGantt1_objectCreated(e);
  }
});
```

In unserem ersten Programmierbeispiel, das die Stufe eins (beabsichtigte Handlungen) verdeutlicht, soll das horizontale Bewegen nur von ganzen Knoten, nicht aber von Teilknoten der Knotengruppe "C" erlaubt sein. Weiterhin soll bei selektierten Verbindungen nur der Quellknoten, nicht aber der Zielknoten veränderbar sein:

```
void jGantt1_prepareModifyInteraction(NeObjectChangeEvent e) throws
                                                        NeVetoException
{

NeIEntity entity = (NeIEntity)e.getObject();
NeObjectChangeInfo theObjectChangeInfo = e.getObjectChangeInfo();

// Horizontales Bewegen von Knoten ist in Gruppe C erlaubt.

boolean moveHorizontal = theObjectChangeInfo.isMoveHorizontal();
if (moveHorizontal && entity.getValueAsString("group").equals("C") &&
    !theObjectChangeInfo.isNodeChange())
  throw new NeVetoException();
```

```
// Nur die Linkquelle lässt sich ändern, nicht das Ziel.
boolean isLinkTargetChange = theObjectChangeInfo.isLinkTargetChange();

if (isLinkTargetChange)
     throw new NeVetoException();
}
```

Auf der gleichen Stufe steht das folgende Beispiel. Vom Programm wird vorgegeben, dass keine Knoten angelegt werden dürfen, deren Starttermin in der Zukunft liegt. Außerdem sollen von Knoten der Gruppe "B" aus keine Verbindungen erzeugt werden dürfen. Dies wird dem Benutzer über eine bestimmte Cursoränderung mitgeteilt und die Anlage des Knotens und der Verbindung werden verhindert:

```
void jGantt1_prepareCreateInteraction(NeObjectChangeEvent e)
                                                    throws NeVetoException
{
NeIEntity entity = (NeIEntity)e.getObject();
NeObjectChangeInfo theObjectChangeInfo = e.getObjectChangeInfo();

// Die Anlage eines Knotens zu einem zukünftigem Datum
// ist nicht möglich und wird z.B. über einen durchgestrichenen
// Anlege-Cursor angezeigt

if (theObjectChangeInfo.isCreateNode() && (new Date
   (theObjectChangeInfo.getDate())).after(new Date
   (System.currentTimeMillis())) ) throw new NeVetoException();

// Das Erzeugen von Links von einem Knoten der Gruppe B ist nicht mög-
lich

if (theObjectChangeInfo.isCreateLink() && entity.getValueAsString
("group").equalsIgnoreCase("B")) throw new NeVetoException();
}
```

Das folgende Beispiel verdeutlicht Stufe zwei. Die Cursoränderung ist erfolgt und ein Phantom wird entweder über die Neuanlage oder die Veränderung eines Knotens bewegt. Die Bewegung ist nur unter bestimmten Bedingungen gestattet: Beim Verschieben darf der Starttermin des bewegten Balkens nicht vor dem 24.2.2008 liegen. Beim Erzeugen eines Knotens darf der Endtermin nicht nach dem 5.5.2008 liegen:

```
void jGantt1_onPhantomModify(NeObjectChangeEvent e) throws NeVetoExcep-
tion
{NeObjectChangeInfo theObjectChangeInfo = e.getObjectChangeInfo();
 Map changes = (Map)e.getValueMap();

// Starttermin nicht vor dem 24.2.
Date refDate = (Date)changes.get("start");
if (refDate != null)
   {  // im Greg. Calender ist Monat 01 = Februar
   if (!theObjectChangeInfo.isNodeChange() && refDate.before(new
       GregorianCalendar(2008, 01, 24).getTime()) )
```

```
        throw new NeVetoException();
    }

// Endtermin eines neu angelegten Knotens nicht später als 5.05.
refDate = (Date)changes.get("end");
if (refDate != null)
    { // im Greg. Calender ist Monat 04 = Mai
      if (theObjectChangeInfo.isCreateNode() && refDate.after(new
      GregorianCalendar(2008, 04, 05).getTime()) )
         throw new NeVetoException();
    }
}
```

Hier ist der Benutzer auf der dritten Stufe angelangt und hat die Maus losgelassen. Das Zurückschreiben der geänderten Daten kann an dieser Stelle noch verhindert werden.

```
void jGantt1_onObjectModify(NeObjectChangeEvent e)
                                                    throws NeVetoException
{NeIEntity ent = e.getEntity();
Date start = ent.getValueAsDate("start");

// Noch sind die Knotendaten nicht zurückgeschrieben, sondern
// ursprünglich und werden daher zunächst gedruckt:
System.out.println ("onObjectModify: "+ start);
…
// Hier wurde die Bool'sche Variable dBQueryObjectModify durch eine
// Datenbankabfrage gesetzt. Ihr Wert entscheidet, ob das
// Zurückschreiben der Daten erfolgen soll oder nicht:
     if (dBQueryObjectModify)throw new NeVetoException();
}
```

Ein ähnliches Fallbeispiel wie oben ließe sich für **onObjectCreate** konstruieren. Mit dem folgenden Beispiel befindet sich der Benutzer auf Stufe vier: Die Datenänderung wurde vorgenommen. Jetzt können nur noch Folgeoperationen verhindert werden.

```
void jGantt1_objectModified(NeObjectChangeEvent e)
{NeIEntity ent = e.getEntity();
 Date start = ent.getValueAsDate("start");

 // Die Daten sind in verändertem Zustand und werden ausgedruckt:
 System.out.println ("onObjectModify: "+ start);
 // Nun erfolgt eine Behandlung, z.B. der Abgleich des Wertes Start
 // und die Durchführung oder Verhinderung eines Datenbankzugriffs.
 …}
```

Ein ähnliches Fallbeispiel ließe sich für **onObjectCreated** konstruieren. Die zur Verfügung stehenden Ereignisse, die Sie in diesem Kapitel kennengelernt haben, bieten Ihnen eine weitestgehende Kontrolle über die Interaktionen des Benutzers und ermöglichen ein fein abgestuftes Eingreifen.

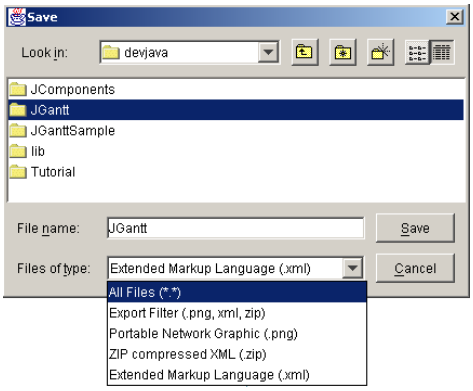
6.3 Diagramm-Schaltleiste

Die Diagramm-Schaltleiste ist eine Werkzeugleiste, die als mögliche Benutzerinteraktionen eine Reihe von Datei- und Druckfunktionen sowie einige Diagrammfunktionen bereitstellt. Die oben abgebildete Leiste zeigt nur die Tasten mit den meist verwendeten Funktionen. Die vollständige Liste der Funktionen finden Sie im Referenzhandbuch.

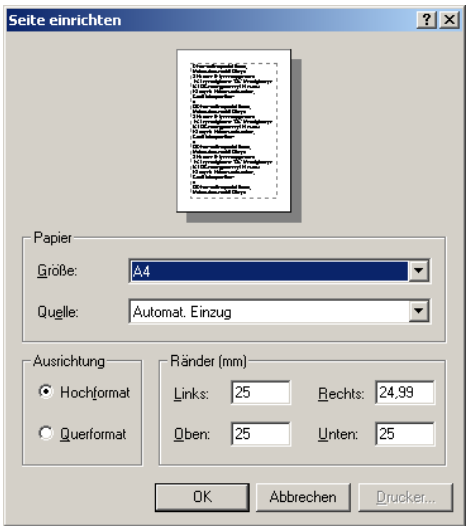
Die Diagramm-Schaltleiste aus Benutzersicht

Bevor wir mit der Programmierung der Schaltleiste beginnen, werfen wir zunächst einen Blick auf die Funktionen der Tasten:

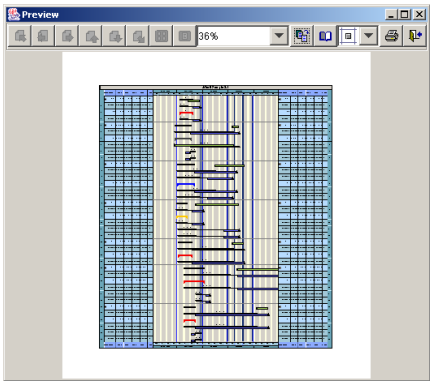
Die Taste mit der Disketten-Ikone speichert oder exportiert die geladene Datei.



Die Taste mit den Buchseiten ruft den Dialog zur Einrichtung der Seite auf.



Die Taste mit der Lupe über der Seite ruft die Seitenvorschau auf.

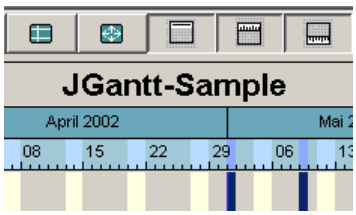


Die Taste mit dem Drucker ruft die Druckfunktion auf.

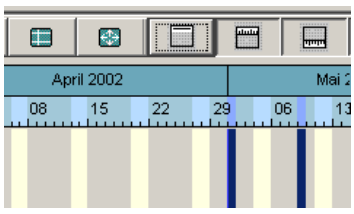
Die Tasten mit den Plus- und Minus-Lupen dienen zum Zoomen des Diagramms. In dem Feld links davon wird der Prozentsatz der Vergrößerung oder Verkleinerung angezeigt und kann editiert werden. Über die 1:1-Taste wird die Grafik in Originalgröße angezeigt.

Mit diesen Tasten wechseln Sie zwischen einer Darstellung mit Teilansichten und der Komplettansicht.

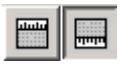
Mit dieser Taste schalten Sie den Diagrammtitel an oder ab.



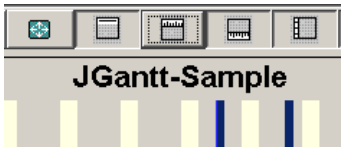
Diagrammtitel ein



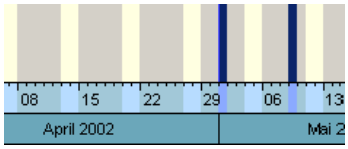
Diagrammtitel aus



Mit diesen Tasten schalten Sie die Zeitskalen an oder aus. Mit der linken Taste setzen Sie die obere, mit der rechten die untere Zeitskala.



Obere Zeitskala aus



Untere Zeitskala ein

Mit diesen Tasten schalten Sie die Tabelle links ein oder aus.

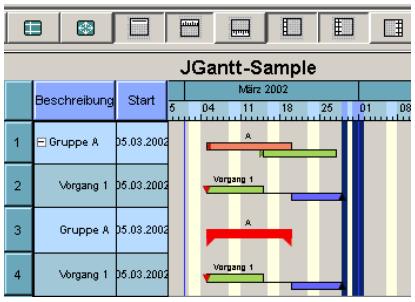
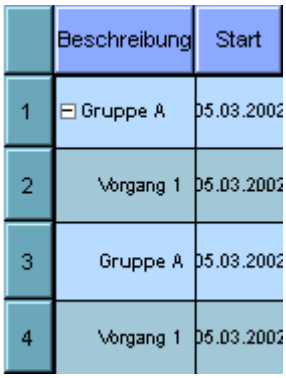


Tabelle linksseitig eingeschaltet

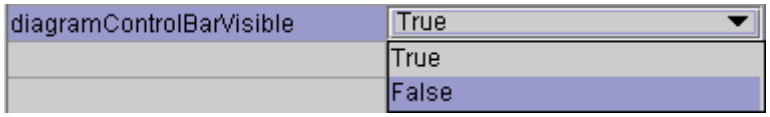
Mit diesen Tasten schalten Sie die Zeilentitel (äußerste Spalte der Tabelle) auf der linken Seite ein oder aus.



Zeilentitel linksseitig eingeschaltet

Die Programmierung der Diagramm-Schaltleiste

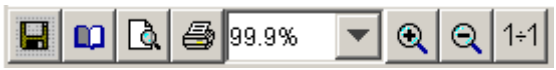
Standardmäßig besitzt ein JGantt-Objekt eine Diagramm-Schaltleiste, die Sie über die Eigenschaften-Tabelle an- oder abschalten können:



… oder über den Quell-Code:

```
// Switching on the ControlPanel in JGantt
jGantt.setDiagramControlBarVisible(true);
```

Standardmäßig ist diese Schaltleiste mit einer begrenzten Anzahl von Tasten ausgestattet:



Die Funktionen Speichern, Seite einrichten, Seitenvorschau und Zoomen sind vorhanden. Falls Sie zusätzliche Tasten anzeigen lassen wollen, fügen Sie diese über die Programmierschnittstelle hinzu:

```
private void setControlPanelButtonsVisible()
{
// Instanz der Schaltleiste vom JGantt-Objekt holen
myControlPanel =    NeDiagramControlPa-
nel)jGantt.getDiagramControlBar();

//Zusätzliche Tasten einfügen
myControlPanel.setButtonVisible
                  (myControlPanel.NE_DIAGRAM_BUTTON,true);
myControlPanel.setButtonVisible
                  (myControlPanel.NE_PANES_BUTTON,true);
myControlPanel.setButtonVisible
                  (myControlPanel.NE_TITLE_BUTTON,true);
myControlPanel.setButtonVisible
                  (myControlPanel.NE_UPPER_TIMESCALE_BUTTON,true);
myControlPanel.setButtonVisible
                  (myControlPanel.NE_LOWER_TIMESCALE_BUTTON,true);
myControlPanel.setButtonVisible
                  (myControlPanel.NE_LEFT_ROW_HEADER_BUTTON,true);
myControlPanel.setButtonVisible
                  (myControlPanel.NE_LEFT_TABLE_BUTTON,true);
}
```

Wenn eine der hinterlegten Eigenschaften gesetzt ist, erscheint die entsprechende Taste automatisch gedrückt. Im Beispiel unten sind Titel und obere Zeitskala sichtbar, sowie die linke Tabelle und die Zeilentitelspalte gesetzt und die entsprechenden Tasten erscheinen automatisch gedrückt:

Nachdem Sie die Schaltleiste auf sichtbar gesetzt haben, werden die Tasten aktiviert:

```
// Aktivierung der Tasten
private void enableControlPanelButtons(boolean bValue)
{
myControlPanel.setButtonEnabled
               (myControlPanel.NE_PANES_BUTTON,bValue);
myControlPanel.setButtonEnabled
               (myControlPanel.NE_DIAGRAM_BUTTON, bValue);
myControlPanel.setButtonEnabled
               (myControlPanel.NE_TITLE_BUTTON, bValue);
myControlPanel.setButtonEnabled
               (myControlPanel.NE_UPPER_TIMESCALE_BUTTON,bValue);
myControlPanel.setButtonEnabled
               (myControlPanel.NE_LOWER_TIMESCALE_BUTTON,bValue);
myControlPanel.setButtonEnabled
               (myControlPanel.NE_LEFT_TABLE_BUTTON,bValue);
myControlPanel.setButtonEnabled
               (myControlPanel.NE_LEFT_ROW_HEADER_BUTTON,bValue);
}
```

Mit der Diagrammschaltleiste verfügen Sie über eine Sammlung von praktischen Funktionalitäten für den Benutzer, die Sie schnell und übersichtlich dem Gantt-Diagramm hinzufügen können.

6.4 Aktionen

VARCHART JGantt bietet die recht einfache Möglichkeit, in der Form von "Aktionen" bestimmte Interaktionen des Benutzers auf Schaltflächen, das Kontext-Menü oder Keyboard-Tasten zu legen. Aktionen in VARCHART JGantt sind eine Erweiterung des Konzeptes der Swing Actions in Java. Sie dienen zur vereinfachten Ansteuerung von Benutzerinteraktionen, die über Menus, Buttons oder Keyboard-Tasten ausgelöst werden sollen. Die durchgeführte Interaktion ist dabei von Ihrer Auslösung durch die GUI-Elemente unabhängig und dadurch leicht handhabbar. Zudem wird die entsprechende Parametrisierung des Buttons, Menus und Tastendrucks an einer Stelle zusammengefasst. Die wichtigsten Aktionen in VARCHART JGantt sind:

- ACTION_SET_SELECT_MODE - Setzen des Auswahlmodus

- ACTION_SET_CREATE_MODE - Setzen des Anlegemodus
- ACTION_DELETE_NODES_AND_LINKS - Löschen von Knoten und Verbindungen
- ACTION_COLLAPSE_GROUP - Gruppe kollabieren
- ACTION_EXPAND_GROUP - Gruppe expandieren
- ACTION_INDENT_NODE - Knoten einrücken
- ACTION_OUTDENT_NODE - eingerückten Knoten herausrücken

In zwei Hinsichten unterscheiden sich die VARCHART JGantt-Aktionen von den Java-Swing Actions:

1. Die Anwendung wird informiert, wenn ein "Auslöser" (z.B. Menüeintrag) für eine Aktion sichtbar gemacht werden soll. Aufgrunddessen kann die Anwendung die geplante Sichtbarmachung verhindern, falls im speziellen Fall gefordert.
2. Die Anwendung wird vor der Ausführung einer Aktion informiert. Auch hier kann die interne Abarbeitung der Aktion verhindert werden, falls gefordert.

Sie haben somit die Möglichkeit, an diesen Stellen in den Programmablauf einzugreifen. Dazu stellt VARCHART JGantt zwei Ereignisse zur Verfügung:

- `onUserActionTriggerShow (NeUserActionEvent e) throws NeVetoException`
- `onUserAction (NeUserActionEvent e) throws NeVetoException`

Für diese Ereignisse können Sie Ihre Applikation beim JGantt-Objekt als Listener registrieren (oder einen angemeldeten Listener löschen):

```
addUserActionListener (NeUserActionListener listener)
removeUserActionListener (NeUserActionListener listener)
```

Das Ereignis **NeUserActionEvent** liefert folgende Informationen, die über get-Methoden abfragbar sind:

- source – den Auslöser des Events
- action – die auslösende Benutzeraktion
- button – den "AbstractButton", falls zutreffend. AbstractButton ist die gemeinsame Oberklasse z.B. von JButton und JMenuItem, d.h. die interaktive Schaltfläche, über die die Benutzeraktion ausgelöst wird.

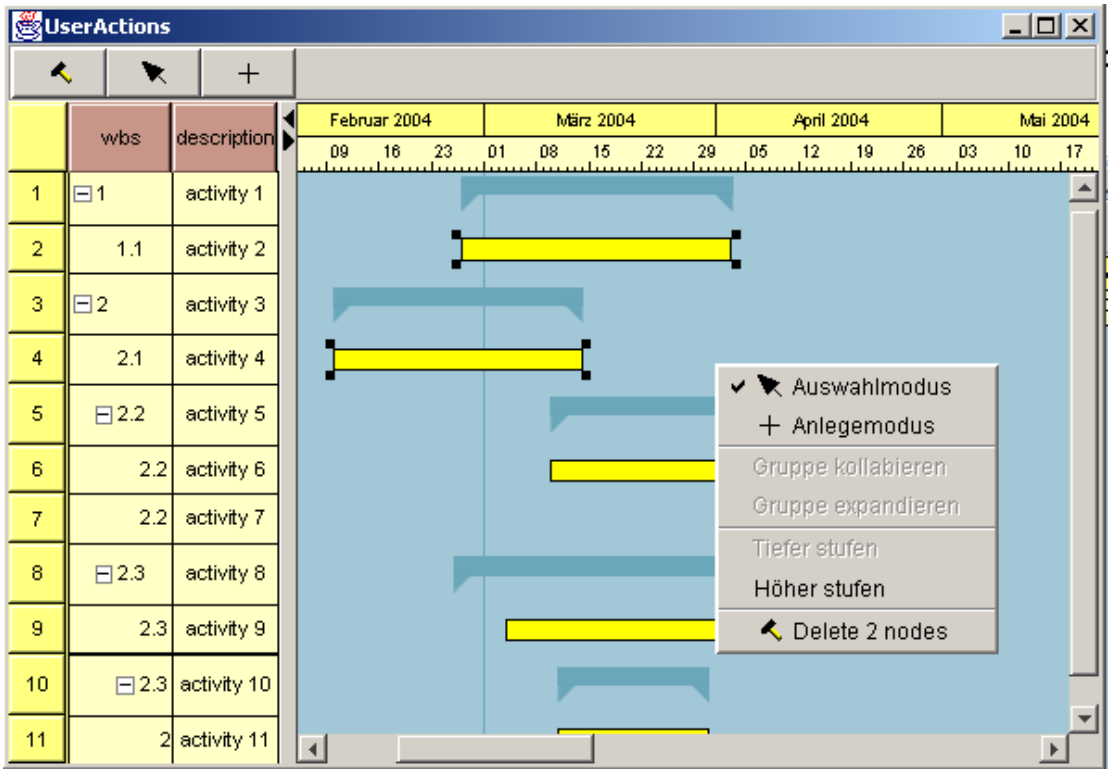
Wie in allen von JComponent abgeleiteten Klassen können auch in VARCHART JGantt Aktionen in Zuordnungstabellen gehalten werden. Hier

werden den Aktionen definierte Namen zugeordnet. Zur Handhabung der Zuordnungstabelle sind in VARCHART JGantt drei Methoden implementiert, die den direkten Zugriff auf die Aktionen ermöglichen:

- `public void removeAction (Object name)`
- `public void putAction (Object name, Action action)`
- `public Action getAction (Object name)`

Über eine weitere Zuordnungstabelle, die sogenannte **inputMap**, wird eine besonders komfortable Verknüpfung zwischen dem Aktionsnamen und den Tastatur-Tasten hergestellt.

Im untenstehenden Beispiel sehen Sie die Aktionen auswählen, anlegen und löschen, die sowohl den Schaltflächen oben links als auch dem Kontextmenü unten rechts zugewiesen wurden. In beiden Fällen wurden auch Ikonen verwendet, die die Interaktionen sowohl auf den Tasten als auch im Kontext-Menü symbolisieren.



Wir werden Ihnen nun die Programmierung und Zuweisung von drei Aktionen mit Hilfe des Beispielprogramms UserActions.java zeigen, das mit dem VARCHART JGantt Software-Paket ausgeliefert wird.

Zunächst benötigen Sie den Zugriff auf eine Reihe von Klassen, um Benutzeraktionen programmieren zu können:

- NeUserAction
- NeUserActionAdapter
- NeUserActionEvent

sowie das Interface

- NeUserActionListener

die Sie am Anfang des Programms importieren:

```
import de.netronic.jgantt.*;
import de.netronic.common.event.*;
import de.netronic.common.intface.*;
import de.netronic.common.beanbase.NeUserAction;
```

Zunächst werden die Instanzen der Aktionen "löschen", "Auswahl-Modus einschalten" und "Anlege-Modus einschalten" vom JGantt-Objekt geholt. Anschließend wird die Lösch-Aktion konfiguriert. Sie soll abgebrochen werden können, wenn sich der Benutzer nach der Sicherheitsabfrage anders entscheidet. Weiterhin darf der Benutzer nicht mehr als 5 Knoten auf einmal löschen, und die Anzahl der zu löschenden Knoten wird im Menü angezeigt:

```
private void initActions()
{
// Aktionen vom JGantt-Objekt holen:
// Aktion wird der Variablen deleteAction zugewiesen
NeUserAction deleteAction =
jGantt1.getUserAction(JGantt.ACTION_DELETE_NODES_AND_LINKS);

// Aktion wird der Variablen selectModeAction zugewiesen
NeUserAction selectModeAction =
jGantt1.getUserAction(JGantt.ACTION_SET_SELECT_MODE);

// Aktion wird der Variablen createModeAction zugewiesen
NeUserAction createModeAction =
jGantt1.getUserAction(JGantt.ACTION_SET_CREATE_MODE);

// Lösch-Aktion konfigurieren
deleteAction.addUserActionListener(new NeUserActionAdapter()
   {
   public void onUserAction(NeUserActionEvent e)
               throws de.netronic.common.event.NeVetoException
     {
     //Aktion wird abgebrochen, falls der Benutzer sich anders
     //entscheidet
     if (JOptionPane.showConfirmDialog(jGantt1, "Are you sure?",
         "Delete selected nodes",
         JOptionPane.YES_NO_OPTION)  ==  JOptionPane.NO_OPTION)
         {
         throw new de.netronic.common.event.NeVetoException ();
         }
     }
```

```
// Die zu löschende Anzahl der Knoten wird ermittelt
   public void onUserActionTriggerShow (NeUserActionEvent e) throws

de.netronic.common.event.NeVetoException
   {
   java.util.Iterator it = jGantt1.selectedEntitiesIterator();
   int i = 0;
   while (it.hasNext())
      {
      it.next();
      i++;
      }

// Der Benutzer darf nicht mehr als fünf Knoten auf einmal löschen
// Dieses Beispiel zeigt die dynamische Änderung eines Menüeintrags

if (i > 5)
   throw new de.netronic.common.event.NeVetoException ();

// Neuen Text für Menüeintrag setzen, der das Ereignis auslöst
e.getButton().setText("Delete " + i + " nodes");
}
  });
```

Nachdem nun die Lösch-Aktion konfiguriert wurde, werden allen drei Aktionen Ikonen zugeordnet und sie werden als Schaltflächen der Werkzeugleiste hinzugefügt:

```
// Ikone wird der Lösch-Aktion zugeordnet:
deleteAction.putValue (Action.SMALL_ICON, new ImageIcon
(useractions.UserActionFrame.class.getResource("hammer.gif")));

// Ikone wird der Anlege-Modus-Aktion zugeordnet:
createModeAction.putValue(Action.SMALL_ICON, new ImageIcon
(useractions.UserActionFrame.class.getResource("create.gif")));

// Ikone wird der Auswahl-Modus-Aktion zugeordnet:
selectModeAction.putValue(Action.SMALL_ICON, new ImageIcon
(useractions.UserActionFrame.class.getResource("select.gif")));

//Zuweisung der Aktionen zu der Werkzeugleiste:
jToolBar.add(deleteAction);
jToolBar.add(selectModeAction);
jToolBar.add(createModeAction);
```

Nachdem nun die drei Aktionen der Werkzeugleiste hinzugefügt wurden, werden die Aktionen **Anlege-Modus** und **Auswahl-Modus** zusätzlich den Keyboard-Tasten "C" und "S" zugewiesen:

```
//Zuweisung der Aktion "Anlege-Modus" zu Taste C:
jGantt1.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put
         (KeyStroke.getKeyStroke("C"), JGantt.ACTION_SET_CREATE_MODE);

//Zuweisung der Aktion "Auswahl-Modus" zu Taste S:
jGantt1.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put
```

```
        (KeyStroke.getKeyStroke("S"), JGantt.ACTION_SET_SELECT_MODE);
```

Sie haben nun über diese Beispiele gelernt, die in VARCHART JGantt vorhandenen Aktionen Keyboard-Tasten und Schaltflächen zuzuweisen, letztere mit Ikonen zu versehen und dynamisch zu gestalten.

Wir werden ihnen jetzt noch ein weiteres Beispiel präsentieren, in welchem Sie für Knotengruppen in der Tabelle beim Expandieren und Kollabieren der Gruppen das +/- Zeichen ausschalten. Dazu registrieren Sie sich, wie bereits gehabt als **UserActionListener** und implementieren die Methode:

```
public void onUserActionTriggerShow (NeUserActionEvent e)
```

Diese Methode wird aufgerufen, bevor VARCHART JGantt ein + oder – Zeichen in die Tabelle zeichnet. Wenn Sie dann eine **NeVetoException** auslösen, erscheint – analog zum Menü-Eintrag – das +/- Zeichen nicht.

Mit dem folgenden Code erlauben Sie das Kollabieren und Expandieren von Gruppen nur auf der obersten Ebene - sowohl über Menu als auch in der Tabelle:

```
...
NeUserActionAdapter collapseExpandAdapter = new NeUserActionAdapter ()
{
public void onUserActionTriggerShow (NeUserActionEvent e) throws
                              de.netronic.common.event.NeVetoException
   {
   // Exception auslösen, wenn getLevel() > 0
   if (getLevel (e.getTarget ()) > 0)
      throw new de.netronic.common.event.NeVetoException ();
   }
 };

jGantt.getUserAction(JGantt.ACTION_EXPAND_GROUP)
                        .addUserActionListener(collapseExpandAdapter);
jGantt.getUserAction(JGantt.ACTION_COLLAPSE_GROUP)
                        .addUserActionListener(collapseExpandAdapter);

// Methode zum Ermitteln der Ebene aus dem Referenzobjekt
   int getLevel (Object ref)
    {
    int level = -1;
    NeILayouterGroup group = null;

//  Das Referenzobjekt kann die Gruppe oder die Gruppen-Entität sein
    if (ref instanceof NeIEntity)
       group = jGantt.getLayouterGroupForNode ((NeIEntity) ref);
    else if (ref instanceof NeILayouterGroup)
       group = (NeILayouterGroup) ref;

    if (group != null)
        level = group.getGroupingLevel ();
    return level;
    }
```

7 Zeitberechnungen

In diesem Kapitel möchten wir Ihnen vorstellen, welche Möglichkeiten VARCHART JGantt hinsichtlich der Zeitberechnung für Projekte bietet.

7.1 Möglichkeiten der Zeitberechung

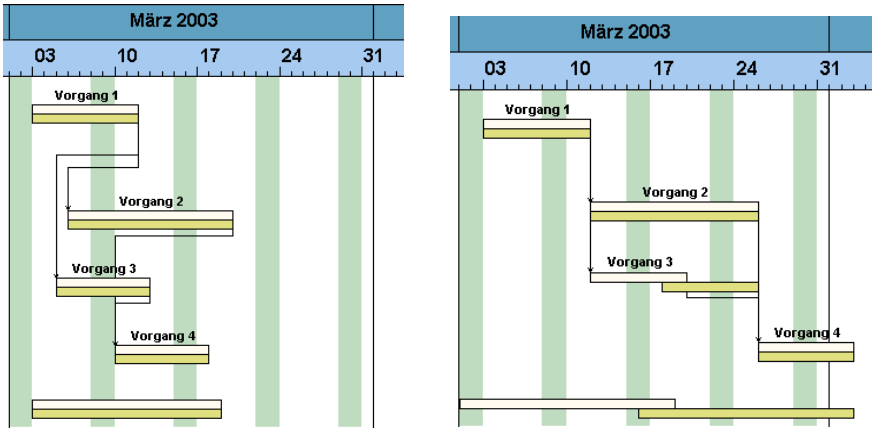
Projekte sind durch einen Anfangs- und einen Endtermin gekennzeichnet, zwischen denen die einzelnen Aktivitäten liegen und innerhalb derer sie in bestimmten Grenzen verschoben werden können.

Die zeitliche Flexibilität einer Aktivität wird durch ihre Abhängigkeit (Verbindung) zu anderen Aktivitäten begrenzt. Diese Flexibilität wird durch die sogenannte Pufferzeit quantifiziert. Es gibt zwei Arten von Pufferzeiten: die freie Pufferzeit und die Gesamtpufferzeit.

Die freie Pufferzeit ist die Zeit, um die eine Aktivität verschoben werden kann, ohne dass andere Aktivitäten davon beeinflusst werden. Die Gesamtpufferzeit ist die Zeit, um die eine Aktivität verschoben werden kann, ohne dass das Projektende sich verschiebt.

Aus den Pufferzeiten ergeben sich für jede Aktivität ein frühestmöglicher Anfang, ein frühestmögliches Ende, ein spätestmöglicher Anfang und ein spätestmögliches Ende.

In den beiden Abbildungen oben sehen Sie links die Anordnung von Knoten vor der Zeitberechnung, rechts nach der Zeitberechnung.



Knotenanordnung vor und nach der Zeitberechnung

Der weiße Layer der Knoten zeigt den jeweils frühestmöglichen Anfang und Ende, der gelbe die spätestmöglichen Termine.

7.2 Einstellungen bei der Zeitberechung

Die Zeitberechnung in VARCHART JGantt führen Sie mit einer eigenen Komponente durch, dem Zeitrechner oder Scheduler. Sie können bei Zeitberechnungen das Anfangsdatum eines Projektes festlegen und das Projektende ermitteln lassen. Damit werden für jede Aktivität die oben genannten Termine berechnet, aus denen sich auch die Pufferzeiten ergeben. Die berechneten Werte werden in dafür vorgesehenen Attributen abgelegt. In Ihrem Projekt vorhandene Profile können wahlweise berücksichtigt werden. Wir zeigen Ihnen nun anhand eines Beispiels, wie Sie eine Zeitberechnung programmieren. Dazu verwenden Sie **ValueReference**-Objekte, die dazu dienen, zusammengesetzte Termine genauso bequem zu handhaben wie einfache und als Übermittler zwischen der Scheduler- und der JGantt-Komponente dienen. Doch zunächst holen Sie ein Scheduler-Objekt:

```
//Vom JGantt-Objekt wird ein Scheduler-Objekt geholt
JGIScheduler myScheduler = myJGantt.getScheduler();
```

Aus der Ableitung des Scheduler-Objektes vom JGantt-Objekt ergibt sich, dass die hier vorhandenen Zuordnungen und Daten dem Scheduler-Objekt bekannt sind. VARCHART JGantt erlaubt aber auch, mit diesem Scheduler die Daten unabhängig von den JGantt-Einstellungen zu berechnen. In einem solchen Fall wäre der folgende Block von Aufrufen notwendig. Für die hier gezeigte Anwendung ist er es nicht, wird aber zu Demonstrationszwecken gezeigt:

```
myScheduler.setAppData(appData());
myScheduler.setCalendar(calendar());
myScheduler.setNodeSetName("Nodes");
myScheduler.setLinkSetName("Links");
myScheduler.setLinkSourceAttribute("sourceNodeUserID");
myScheduler.setLinkTargetAttribute("targetNodeUserID");
myScheduler.setLinkTypAttribute("type");
```

Jetzt legen Sie die Value-Reference-Objekte an. Dabei werden diesen die Attributnamen der Entitäten mitgeteilt, in die vom Scheduler berechneten Werte eingeschrieben werden. Zunächst werden zwei ValueReference-Objekte angelegt, die als Input-Parameter einer Dauer bei Knoten und bei Verbindungen dienen. Ihre Werte werden gleich in Millisekunden umgewandelt:

```
// ValueReference durationNode korrespondiert mit Attribut "duration"
NeValueReference durationNode = new NeValueReference("duration");

//Umrechnung in Millisekunden
durationNode.setFactor(NeValueReference.DURATION_FACTOR_DAY);

// ValueReference durationLink korrespondiert mit Attribut "duration"
```

```
NeValueReference durationLink = new NeValueReference("duration");

//Umrechnung in Millisekunden
durationLink.setFactor(NeValueReference.DURATION_FACTOR_DAY);
```

Jetzt werden der frühestmögliche Anfang und das Ende, sowie der spätestmögliche Anfang und das Ende angelegt:

```
// ValueReference earlyStart korrespondiert mit Attribut "start"
NeValueReference earlyStart = new NeValueReference("start");

// ValueReference earlyEnd korrespondiert mit Attribut "end"
NeValueReference earlyEnd = new NeValueReference ("end");

// ValueReference lateStart korrespondiert mit Attribut "start_1"
NeValueReference lateStart = new NeValueReference("start_1");

// ValueReference lateEnd korrespondiert mit Attribut "end_1"
NeValueReference lateEnd = new NeValueReference("end_1");
```

Zu den Terminen einer Aktivität sollen auch noch die freie und die Gesamtpufferzeit ermittelt werden. Ihre ValueReference-Objekte werden hier angelegt:

```
// ValueReference freeFloat korrespondiert mit Attribut "f_float"
NeValueReference freeFloat = new NeValueReference("f_float");

// ValueReference totalFloat korrespondiert mit Attribut "t_float"
NeValueReference totalFloat = new NeValueReference("t-float");
```

Nun werden die ValueReferenz-Objekte an den Scheduler überwiesen:

```
myScheduler.setEarlyStartReference(earlyStart);
myScheduler.setEarlyEndReference(earlyEnd);
myScheduler.setLateStartReference(lateStart);
myScheduler.setLateEndReference(lateEnd);
myScheduler.setFreeFloatReference(freeFloat);
myScheduler.setTotalFloatReference(totalFloat);
myScheduler.setNodeDurationReference(durationNode);
myScheduler.setLinkDurationReference(durationLink);
```

Bevor Sie nun die Berechnung durchführen, setzen Sie noch die Profile, die bei der Berechung berücksichtigt werden sollen:

```
myScheduler.setNodeProfileAttribute(profile);
myScheduler.setLinkProfileAttribute(profile);
```

Jetzt setzen Sie den Projektanfang, um das Ende berechnen zu lassen:

```
myScheduler.setProjectStart(projectStart);
```

Dabei wird zunächst eine sogenannte "Vorwärtsrechnung" durchgeführt, bei der die frühesten Termine berechnet werden. Anschließend wird die Rückwärtsrechnung durchgeführt, bei der die spätesten Termine berechnet werden.

Sie könnten entsprechend auch das Projektende setzen und dann den Anfang berechnen lassen. Dabei wird zuerst die Rückwärtsrechnung, dann die Vorwärtsrechnung durchgeführt.

Sie könnten zudem sowohl den Projektanfang als auch das Ende setzen, wobei sich die Pufferzeiten entsprechend ausdehnen oder verkürzen würden. Bei zu geringer Zeitmenge nehmen die Pufferzeiten negative Werte an. Jetzt wird die Berechnung durchgeführt und die Werte in die Attribute eingeschrieben:

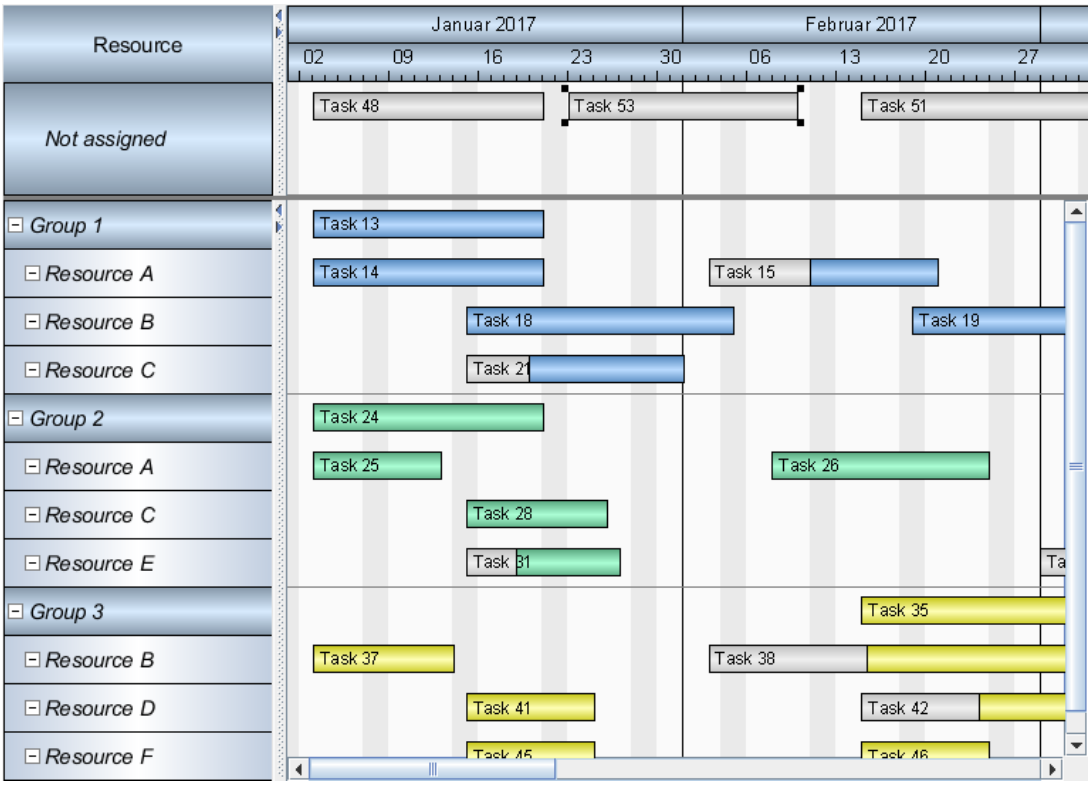
```
myScheduler.schedule();
```

Die Vorstellung der Zeitberechnung ist damit abgeschlossen.

8 Das Synchronizer-Panel

Eine häufige Anforderung an Applikationen ist es, zwei Gantt-Diagramme synchron darzustellen, in welchem das eine ein Lager mit inaktiven Vorgängen vorhält, während im zweiten die Vorgänge in Aktion gezeigt werden.

Typisch sind solche Darstellungen z.B. in Anwendungen des Bereichs Logistik, wo in einem Fahrzeug-Pool parkende und arbeitende Fahrzeuge abgebildet werden, oder in Anwendungen von Produktionsvorgängen, in welchen mit Werkzeugen bestückte oder unbestückte Maschinen visualisiert werden.



Im Bild oben sehen Sie oberhalb der grauen Linie ein Lager mit inaktiven, hellgrauen Vorgängen (Tasks not assigned), während sich unterhalb die aktiven Vorgänge (farbige Tasks) befinden. Die inaktiven Vorgänge oberhalb der grauen Linie können per Drag&Drop in das aktive Diagramm nach unten gezogen und so in den laufenden Betrieb eingeordnet werden.

Eine derartige Darstellung ist besonders gut geeignet für sehr große Diagramme, da das aktive Diagramm Rollbalken enthält und deshalb die Entfernungen bei Drag&Drop-Aktionen durch eine vorteilhafte Einstellung der Rollbalken verkürzt werden, was den Verschiebevorgang sehr komfortabel gestaltet.

Für eine derartige synchrone Darstellung zweier Gantt-Diagramme verwenden Sie die JGantt-Klasse „Synchronizer-Panel“.

Beim Anlegen eines Synchronizer-Panels werden automatisch zwei JGantt-Instanzen definiert, die voneinander abhängig sind. Die obere Instanz ist durch ein „Slave“-Verhältnis zu der unteren „Master“-Instanz definiert:

```
import de.netronic.jgantt.JGanttSynchronizerPanel;

JGanttSynchronizerPanel jSynchPanel =
                                new JGanttSynchronizerPanel (slaveFilter);
```

Durch diesen Aufruf legen Sie mit Hilfe des Konstruktors ein Synchronizer-Panel an, das eine Master- und eine Slave-JGantt-Instanz enthält. Die beiden Instanzen teilen automatisch alle Elemente und Eigenschaften und greifen auch auf dieselbe AppData zu.

Um jedoch Knoten separat auf die beiden Instanzen verteilen zu können, legen Sie einen slaveFilter an. Erfüllt ein Knoten die Bedingung des Filters, wird er der Slave-Instanz zugewiesen; erfüllt er sie nicht, bleibt er in der Master-Instanz.

```
NeIFilter slaveFilter = new NeFilterBase ()
{public boolean apply(NeIEntity theEntity)
 {return theEntity.getValueAsString("Group").startsWith("Not assigned");
 }
};
```

Die Knoten werden auf diese Weise disjunkt auf die Instanzen verteilt: Slave- und Master-Instanz besitzen keine gemeinsamen Knoten, zudem werden ausnahmslos alle Knoten auf die Instanzen verteilt.

Die Synchronisation von Master- und Slave-Instanz muss von Ihnen aktiv eingeschaltet werden. Dabei werden alle Eigenschaften von der Master-Instanz auf die Slave-Instanz kopiert. Dadurch haben Sie die Möglichkeit, nach der Synchronisierung noch Unterschiede im Aussehen oder im Verhalten zu programmieren. Zum Synchronisieren setzen Sie die Panel-Eigenschaft „active“ auf true:

```
//Synchronisation
jSynchPanel.synchronize();
```

Anschließend können Sie z.B. das Gruppen-Layout oder das Knotenverhalten in der einen oder der anderen Instanz neu konfigurieren:

```
// Setzung eines anderen GroupLayouts in der Slave-Instanz
jSynchPanel.getSlave().setGroupLayout
                                (JGantt.GROUP_ROWLAYOUT_SINGLE_OPTIMIZED);
```

```
// Setzung eines anderen Modus des KnotenResizing in der Slave-Instanz
jSynchPanel.getSlave().setInteractionNodeResizeMode
                              (JGantt.INTERACTION_RESIZE_MODE_LEFT_RIGHT);
```

Auf diese Weise können Sie die beiden JGantt-Instanzen überwiegend gleich (synchron), aber dennoch mit Unterschieden einrichten.

In der mitgelieferten Beispielsammlung finden Sie das Programm **ControlCenter.java**, das die Anwendung des Synchronizer-Panels exemplarisch vorführt.

9 Der Persistenz-Manager

Der Persistenz-Manager ist ein Interface der JGantt-Komponente, das den aktuellen Zustand der Objekte in der JGantt-Komponente speichert und es ermöglicht, diesen Zustand später wieder zu laden.

So gehen Sie dabei vor:

```
import de.netronic.jgantt.JGPersistenceManager;
...
// Setzungen für die Persistence
      JGIPersistenceManager persMan1 = jGantt1.getPersistenceManager();
      persMan1.setEditingSwitchInPropertyBrowserEnabled(true);
      persMan1.setDocPath("file:../../doc/");
```

10 Der Eigenschaften-Browser

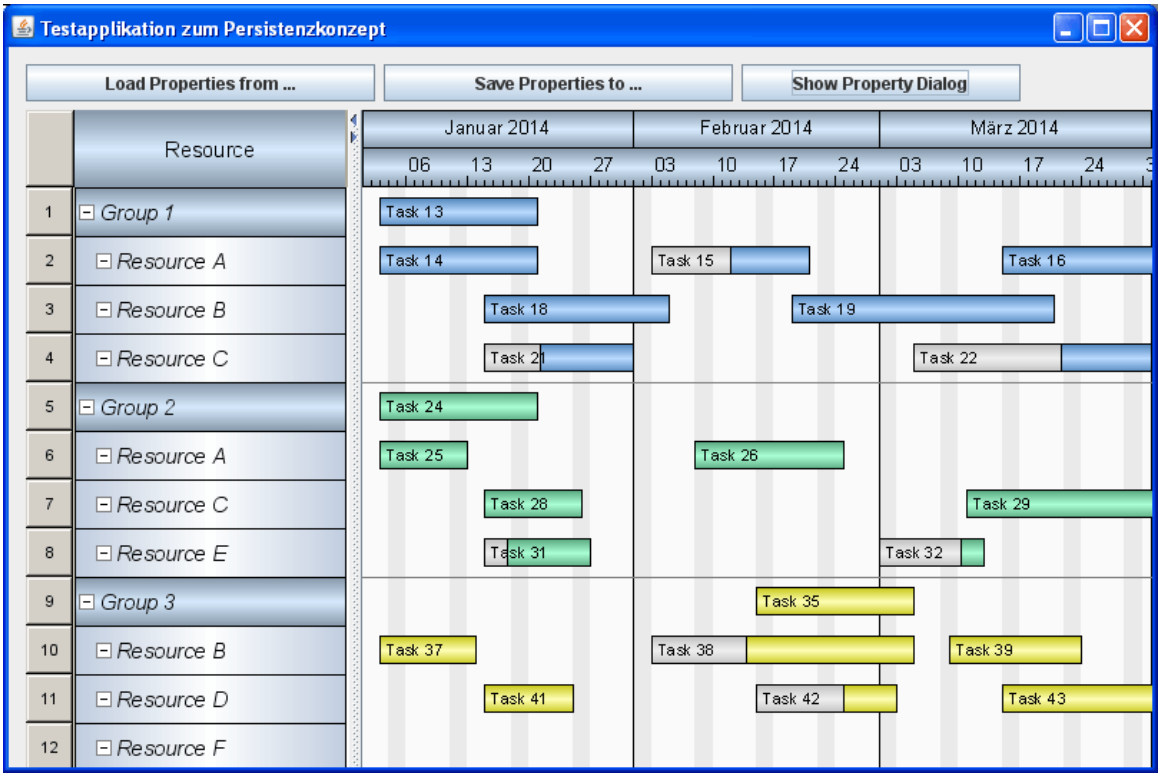
Der Eigenschaften-Browser ist ein Dialog, der speziell für das komfortable Ausprobieren von Eigenschaften in VARCHART JGantt entwickelt wurde. Sie können damit zu beliebigen Zeitpunkten Applikations- und Kalenderdaten sowie alle Eigenschaften und deren Setzungen im XML-Format einsehen.

Der Eigenschaften-Browser kann über die API oder über die Tastatur mit Strg+Umschalt+F11aktiviert werden. Mit Ausnahme der Nur-Lese-Eigenschaften lassen sich alle Daten mit diesem Browser auch ändern.

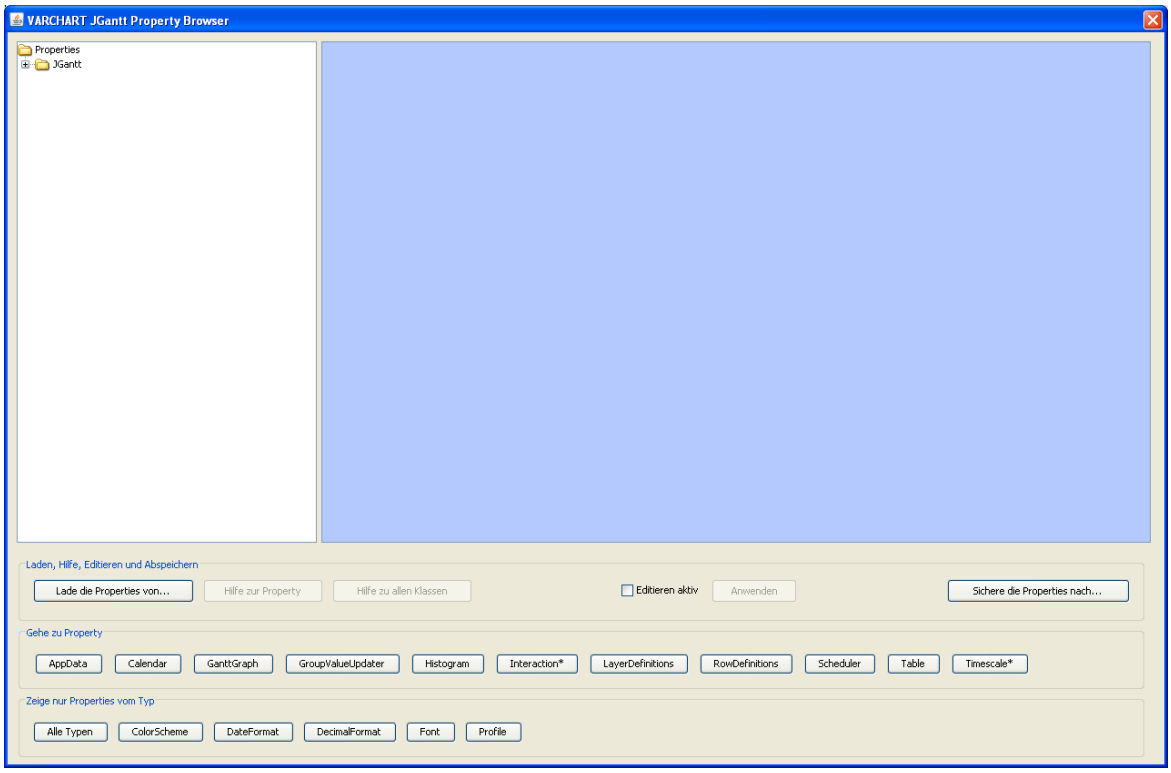
Der Browser verschafft Ihnen einen guten Überblick über den aktuellen Zustand der Einstellungen Ihres Gantt-Diagramms. Das Auffinden entscheidender Stellen erfolgt beschleunigt. Sie können sehr sicher und schnell durch Ihre Einstellungen navigieren und Werte, die Sie gesetzt haben, einsehen und ändern.

Der Brower eignet sich besonders dazu, Varianten Ihres Gantt-Diagramms auszuprobieren. Sehr schnell können Sie hiermit Design-Optionen von Objekten ändern und die jeweilige Version Ihres Diagramms in einer eigenen XML-Datei ablegen. Das Demonstrieren verschiedener Varianten Ihres Gantt-Diagramms wird damit sehr komfortabel.

Im Bild unten wurde aus einer Anwendung heraus der Browser aktiviert und eine XML-Datei geladen (Schaltfläche **Load properties from....**)

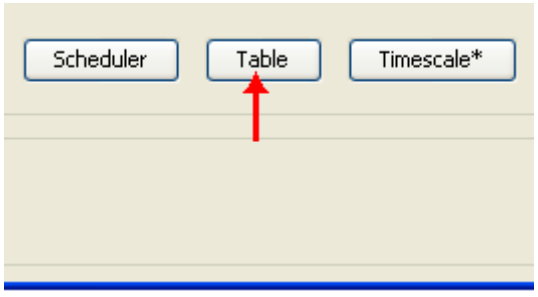


Wenn Sie nun die Schaltfläche **Show property dialog** anklicken, erscheint der entsprechende Dialog:

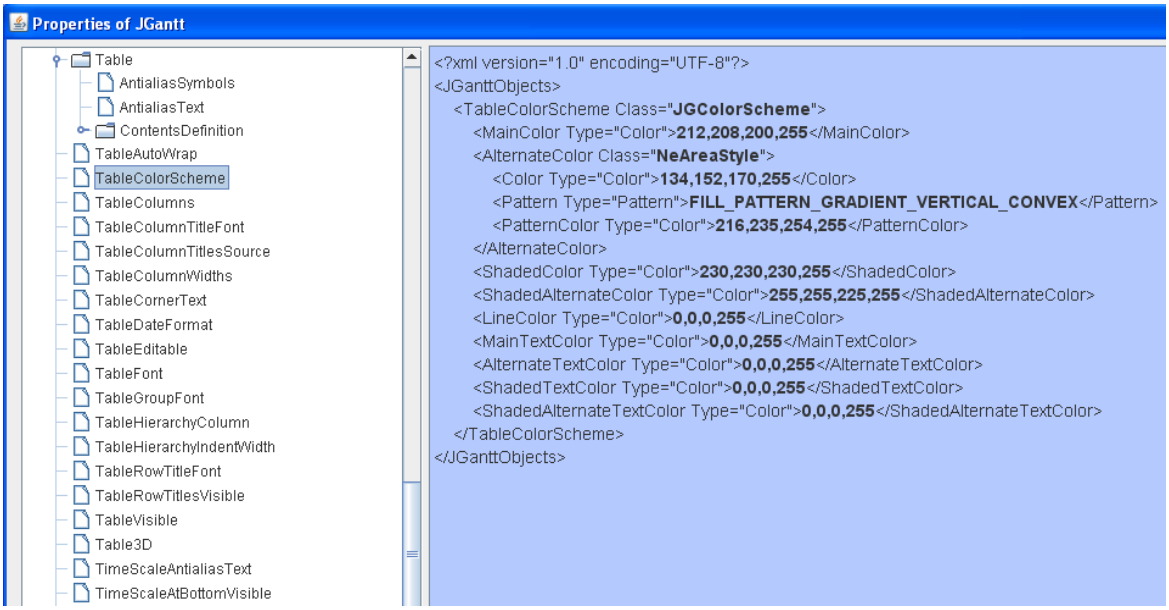


Sie sehen im linken Fenster die Darstellung der (zusammengefalteten) JGantt-Komponente, das rechte Fenster ist leer und unterhalb der beiden Fenster Leisten mit Schaltflächen.

Durch Anklicken einer der Schaltflächen, z.B. „Table“ erfolgt das Auflklappen der JGantt-Komponente und die automatische Navigation zu den Setzungen aller Tabellenelemente im linken weißen Fenster:



Wenn Sie nun das Objekt im weißen Fenster anklicken, das Sie verändern wollen – z.B. TableColorScheme - erscheinen deren Eigenschaften und Setzungen im rechten blauen Fenster:



Dort können Sie nun beliebig editieren. Wenn Sie neue Setzungen ausprobieren oder speichern wollen, drücken Sie die Schaltflächen **Apply** bzw. **Save properties to**. Mit letzterer Schalfläche können Sie also beliebig viele Varianten Ihres Gantt-Diagramms anfertigen und für Demonstrationszwecke speichern.

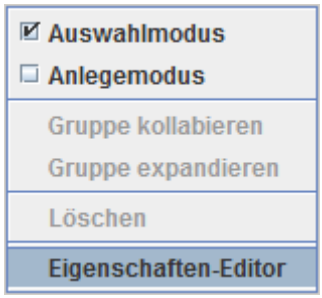
Sie können mit diesem Tool Ihre Ideen schnell umsetzen und das Ergebnis leicht demonstrieren. Der Eigenschaften-Browser vereinfacht und beschleunigt auf diese Weise die Kommunikation mit Ihrem Kunden oder Auftraggeber.

11 Der Eigenschaften-Editor

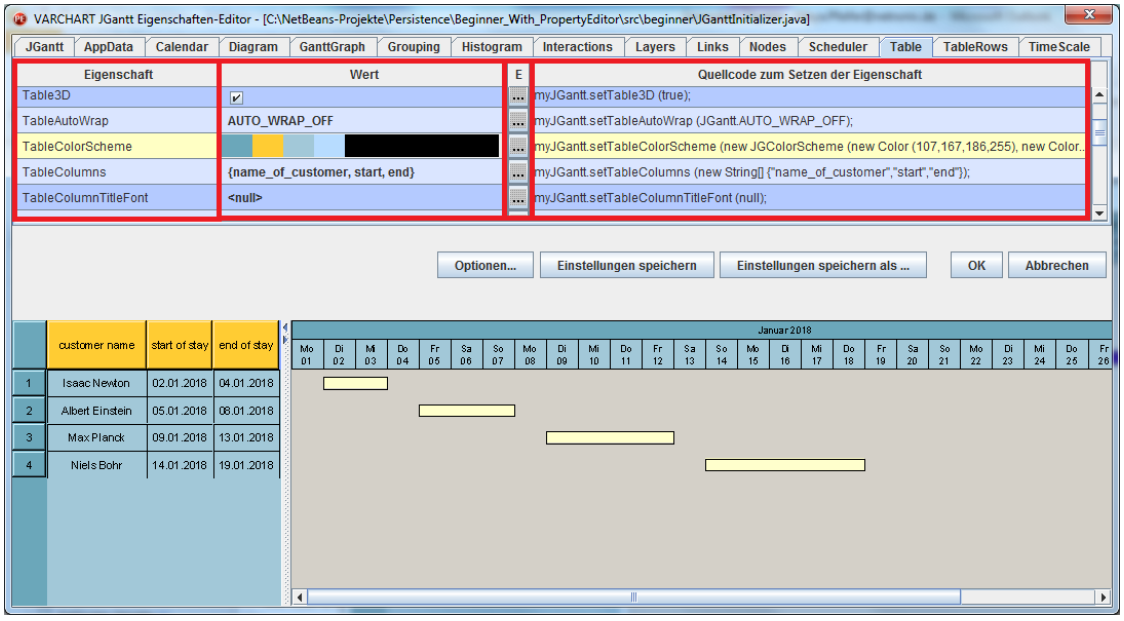
11.1 Grundlegende Funktionen

Der Eigenschaften-Editor bietet Ihnen Unterstützung beim Erstellen einer neuen JGantt-Applikation. Sie können mit diesem Werkzeug Ihre Diagramme sehr komfortabel entwickeln und konfigurieren.

Um den Eigenschaften-Editor nutzen zu können, muss dieser im Quell-Code aktiviert werden; wenn dies erfolgt ist, können Sie dieses Werkzeug mit der Tastenkombination <Strg> <Umschalt> <F10> in jeder Anwendung aufrufen oder einfacher, über das im Gantt-Graphen aufrufbare Kontext-Menü (rechte Maustaste).



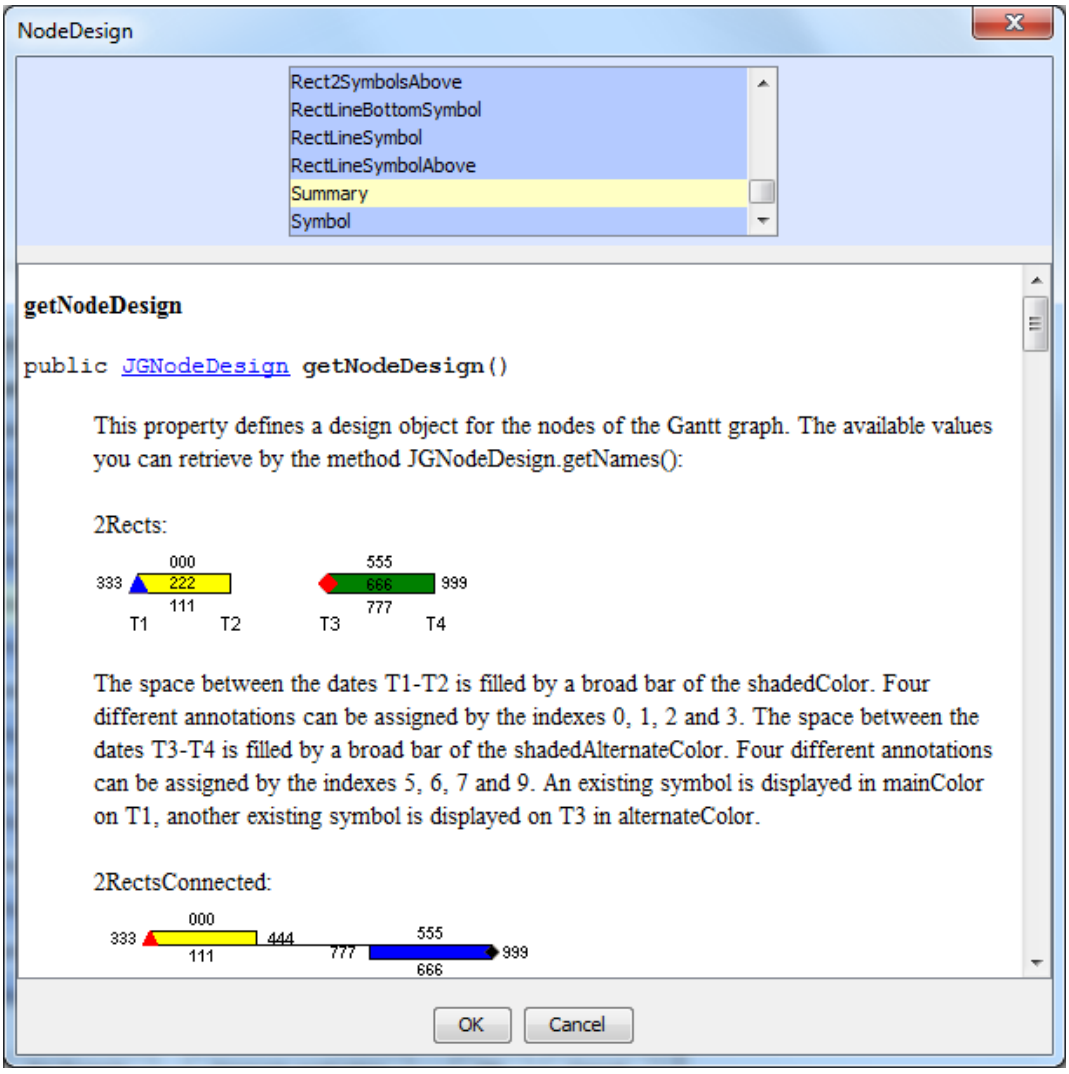
Wenn Sie den Eigenschaften-Editor aufrufen, sehen Sie am oberen Rand eine Reihe von Registern, auf denen alle Eigenschaften von VARCHART JGantt nach inhaltlichen Kategorien angeordnet sind. Sobald Sie ein Register anklicken, werden die Eigenschaften der zugehörigen Objekte darunter in einer Tabelle aufgelistet:



Die rot gerahmte Liste im Bild oben zeigt die Eigenschaften des aktivierten **Table**-Registers. In der Liste sehen Sie vier Spalten. Die erste Spalte am linken Rand unter dem Titel „Eigenschaft“ zeigt die Namen der Eigenschaften in alphabetischer Reihenfolge an. Da die Anzahl der Eigenschaften beim Objekt „Table“ sehr umfangreich ist, besitzt die Liste am rechten Bildrand einen Rollbalken.

Rechts von der Spalte **Eigenschaft** finden Sie die Spalte **Wert**. Hier wird der Wert angezeigt, der aktuell der Eigenschaft zugeordnet ist. Dieses Feld können Sie meist direkt (in-place) editieren und somit Änderungen vornehmen; wo dies nicht möglich ist, wird ihnen eine vorgegebene Werteauswahl angeboten.

Rechts davon sehen Sie eine Schaltfläche mit drei Punkten: … Hier ist erweitertes Editieren (E) z.B. über einen Dialog möglich. Unten sehen Sie beispielhaft einen derartigen Dialog, hier zur Eigenschaft **Knotendesign** des Registers **Node**.



Während Sie im oberen Teil des Dialogs ein Knotendesign für einen Knoten auswählen können, sehen Sie im unteren die Deklaration der Eigenschaft sowie Erläuterungen der unterschiedlichen Designs mittels Text und Bild.

Im Eigenschaften-Editor (s. Bild vorhergehende Seite) sehen Sie rechts von der Spalte **Wert** die Spalte **Quellcode zum Setzen der Eigenschaft**. Dort wird Ihnen die Code-Zeile gezeigt, mit der Sie die Eigenschaft wie gewünscht setzen könnten.

Unterhalb der Auflistung sehen Sie ein Vorschau-Gantt-Diagramm, in welchem die bestehenden Setzungen und die von Ihnen vorgenommenen Änderungen sichtbar sind bzw. werden.

Auf diese Weise können Sie alle Eigenschaften der JGantt-Bean sehr komfortabel setzen.

11.2 Anwendungsszenarien

11.2.1 Das Kennenlernen des Eigenschaften-Editors

Um sich mit der Umgangsweise des Eigenschaften-Editors vertraut zu machen, empfehlen wir, mit dem Quellcode-Beispiel **Beginner** anzufangen.

Dazu sollten Sie, nachdem Sie VARCHART JGantt installiert haben, das Quellcode-Verzeichnis des Beginner-Projektes mit dem Namen **src** in ein neues Projekt in der eigenen Entwicklungsumgebung kopieren. Anschließend sollten Sie die Funktionsbibliothek JGantt.jar dem Projekt bekannt machen. Dann öffnen Sie die Datei **Beginner.java**.

Suchen Sie hier zunächst im Quellcode den Aufruf

```
myJGantt.getPropertyEditor().setEnabled(true);
```

Hier können Sie die Möglichkeit, den Eigenschaften-Editor aufzurufen, an- oder abschalten. Vergewissern Sie sich, dass der Wert auf **true** steht. Nun übersetzen Sie das Programm und führen es aus. Er erscheint ein einfaches Gantt-Diagramm.

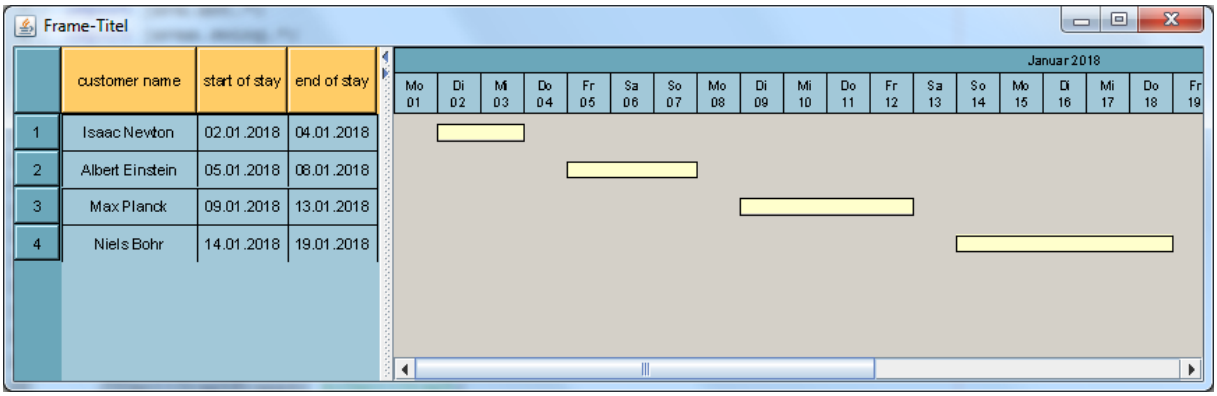
Nachdem Sie einmal in das Diagramm geklickt haben, können Sie mit der Tastenkombination <Strg> <Umschalt> <F10> oder über das Kontext-Menü den Eigenschaften-Editor aufrufen (diese Möglichkeiten werden abgeschaltet, sobald Sie die **Enabled**-Eigenschaft auf **false** setzen).

Wenn Sie sich den Quellcode von **Beginner** ansehen, werden Sie die JGantt-Initializer-Klasse sehen, über die alle JGantt-Eigenschaften und –Entitäten initialisiert werden.

In der Methode **jbInit** wird nach der Anlage einer JGantt-Instanz zunächst die Initialisierung aufgerufen:

```
myJGantt = new JGantt ();
new JGanttInitializer(myJGantt).initJGantt();
```

Das so initialisierte Gantt-Diagramm können Sie nun ausführen und sehen vier Knoten mit hellgelben Layern:


Frame-Titel
customer name
start of stay
end of stay
1 Isaac Newton 02.01.2018 04.01.2018
2 Albert Einstein 05.01.2018 08.01.2018
3 Max Planck 09.01.2018 13.01.2018
4 Niels Bohr 14.01.2018 19.01.2018
Januar 2018
Mo 01 Di 02 Mi 03 Do 04 Fr 05 Sa 06 So 07 Mo 08 Di 09 Mi 10 Do 11 Fr 12 Sa 13 So 14 Mo 15 Di 16 Mi 17 Do 18 Fr 19

Sie können nun, wie im vorangegangenen Kapitel beschrieben, die Konfiguration dieses Diagramms über die Register und Eigenschaften vornehmen, beispielsweise, indem Sie über das Register **Layer** die Eigenschaft **AreaStyle** des Layers **layer_node_0** verändern (d.h. die Farbe des Layers).

Wenn Sie anschließend die Schaltfläche **Einstellungen speichern** drücken, werden Ihre Änderungen in die Klasse **JGanttInitializer** zurückgeschrieben. Nach erneutem Kompilieren werden die Änderungen in der Anwendung wirksam.

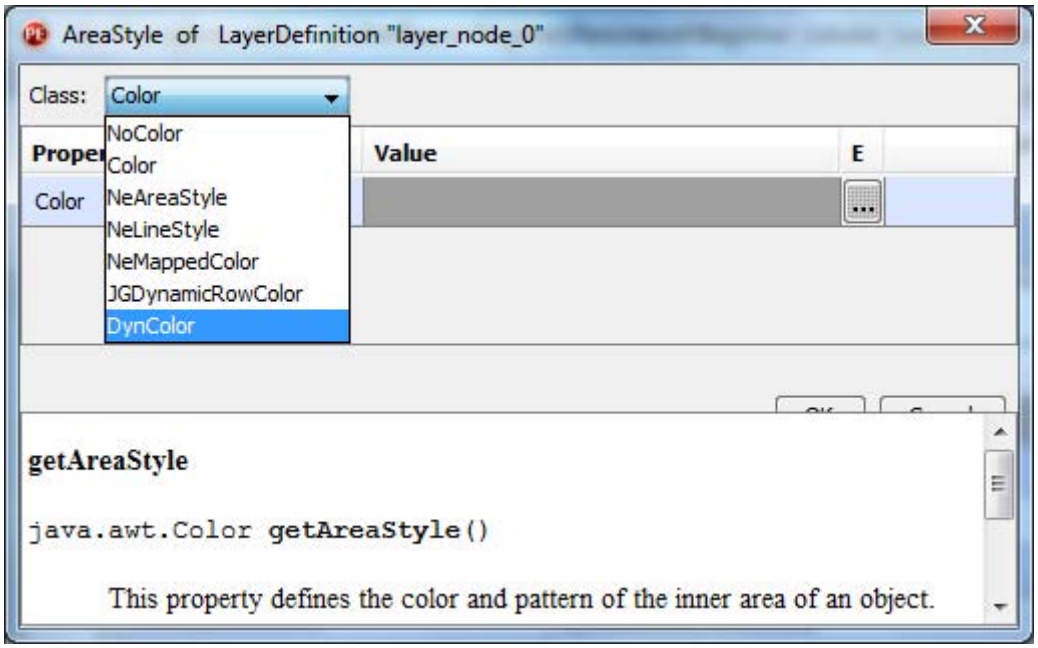
11.2.2 Die Verwendung selbstgeschriebener Klassen

Sie können in dem Eigenschaften-Editor auch selbstgeschriebene Klassen verwenden. Wir zeigen dies am Beispiel der selbstgeschriebenen Klasse **DynColor**, die die Interfaces **NeIDynamicColor** und **NeISourceString** implementiert.

Dank der Implementation des Interfaces **NeISourceString** kann diese dynamische Farbe im Eigenschaften-Editor benutzt werden:

```
// Initialisierung der JGantt-Instanz und Bekanntgabe d. Klasse DynColor
myJGantt = new JGantt ();
myJGantt.getPropertyEditor().addNeIDynamicColor(new DynColor())
```

Sie können die Applikation nun ausführen, in das Diagramm klicken und den Eigenschaften-Editor aufrufen. Hier bietet nun das Register **Layer** an, bei der Eigenschaft **AreaStyle** des Layers **layer_node_0** die dynamische Farbe dem Layer zuzuweisen, indem Sie in der Combobox die Klasse **DynColor** auswählen.



Klicken Sie im AreaStyle- und den Layer-Dialog jeweils auf **OK** und auf der Oberfläche des Eigenschaften-Editors auf **Einstellungen speichern**. Damit haben Sie die Zuweisung der dynamischen Farben nun im Quell-Code gesichert.

11.2.3 Übertragen von Testeinstellungen auf Produktivdaten

Wenn Sie mit Hilfe des Beginner-Beispiels Einstellungen vorgenommen haben und diese Einstellungen auf „echte“ Produktivdaten übertragen wollen, sollten Sie die JGantt-Bean initialisieren, ohne zunächst die AppData-Entitäten zu laden. Dazu gibt es eine entsprechende Methode:

```
new JGanttInitializer(myJGantt).initJGanttWithoutAppDataEntities();
```

Nun können Sie Ihre Produktivdaten in die AppData laden und damit die JGanttBean mit Ihren Testeinstellungen und den Produktivdaten initialisieren.

11.2.4 Neubeginn einer Applikation

Sie können auch mit dem Eigenschaften-Editor eine Applikation komplett neu starten. Zuerst definieren Sie das Package, in welches Sie Ihre Klassen legen wollen, da Sie kein Default-Package benutzen dürfen.

Nachdem Sie also in Ihrer Programmierumgebung aus der Angebotspalette möglicher Beans die VARCHART JGantt-Bean auf ein JFrame gezogen haben (womit die JGantt-Bibliothek automatisch in das aktuelle Projekt eingefügt wird), fügen Sie den folgenden Konstruktor in Ihren Quell-Code ein:

```
public MyFrame()
    {
    initComponents();

    // GanttGraphen mit leerer Zeile generieren
    jGantt1.setDiagramEmptyRow(true);

    //Eigenschaften-Editor aktivieren
    jGantt1.getPropertyEditor().setEnabled(true);

   }
```

Anschließend starten Sie die Applikation. Es erscheint ein leeres Diagramm mit Zeitskala, Tabelle und Gantt-Graph, aber ohne Knoten.

Um Einstellungen vornehmen zu können, sollten Sie nun sinnvollerweise ein paar Knoten generieren. Den Anlegemodus finden Sie im Kontext-Menü, das Sie mit der rechten Maustaste aufrufen können.

Anschließend rufen Sie mit <Strg> <Umschalt> <F10> den Eigenschaften-Editor auf und konfigurieren das Grunddiagramm entsprechend Ihren Vorstellungen. Mit der Schaltfläche **Einstellungen speichern unter** erstellen Sie im Quellcode-Verzeichnis der Anwendung eine Klasse mit dem Namen **JGanttInitializer** (der Name wird automatisch vorgeschlagen).

Generieren Sie nun zunächst eine neue JGantt-Instanz:

```
<Name der JGantt-Variable> = new JGantt ();
```

Anschließend vererben Sie dieser die Initialisierungen des bestehenden Gantt-Diagramms:

```
new JGanttInitializer(<Name der JGantt-Variable>).initJGantt();
```

11.2.5 Einen Schnappschuss machen

Es ist sinnvoll, Ihre Änderungen als neue Anwendung zu speichern, beispielsweise wenn Sie Ihrem Kunden eine Auswahl derselben Anwendungen etwa mit Farb- oder Formvarianten von Knoten zeigen wollen. Dazu können

Sie eine Serie von Quellcode-Beispielen („Schnappschüsse“) abspeichern, die sich anschließend gut vorführen lassen.

Sie wollen also einen bestimmten Zustand Ihrer Applikation als eigenständige Datei abspeichern, anschließend aber an Ihrer ursprünglichen Applikation weiterentwickeln. Dafür muss zu jedem „Schnappschuss“ auch eine **main**-Methode mitgespeichert werden. Dazu gehen Sie wie folgt vor:

Sie rufen aus Ihrer Applikation heraus mit <Strg> <Umschalt> <F10> oder über das Kontext-Menü den Eigenschaften-Editor auf. Mit dem Anklicken der Schaltfläche **Optionen** erreichen Sie den Optionen-Dialog, aktivieren darin die Checkbox **Generierten Code als Applikation mit „main“-Methode speichern** und speichern mit **OK** ab. In der Oberfläche des Eigenschaften-Editors drücken Sie die Schaltfläche **Einstellungen speichern als** und speichern die erzeugte Klasse (den Schnappschuss) in das Quellcodeverzeichnis, wo auch die ursprüngliche Klasse mit Main-Methode liegt (achten Sie bitte dabei darauf, dass Sie keine bestehende Klasse, z.B. JGanttInitializer.java, überschreiben).
Sie können anschließend die Schnappschuss-Applikation mit **Run File** laufen lassen.

Sollten Sie eigene Filter und Farben geschrieben haben, sehen Sie dazu wieder Kap. 1.2.2)

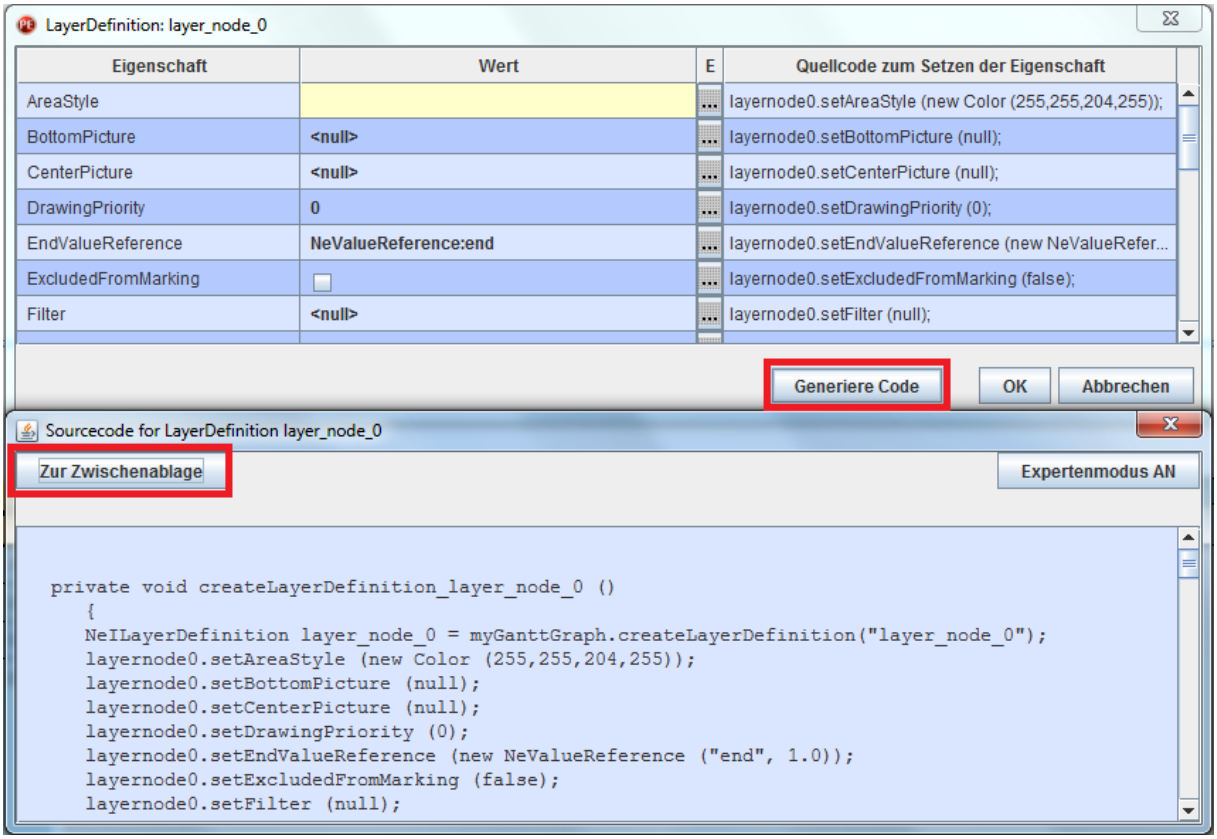
11.2.6 Eigenschaften übertragen

Wenn Sie Änderungen an der Konfiguration eines Testdiagramms machen und den Quellcode dieser Änderungen in Ihre Produktivapplikation übertragen wollen, nehmen Sie dort einen „Quellcode-Einschub“ vor.

Nehmen wir an, Sie ändern in Ihrem Testdiagramm mit Hilfe des Eigenschaften-Editors beispielsweise eine Layer-Definition (Register **Layers**).

Nun wollen Sie die Konfiguration dieser Eigenschaft als Quellcode in Ihre Produktivapplikation übertragen

Im Dialogfenster der LayerDefinition drücken Sie die Schaltfläche **Code generieren**.



Der generierte Code wird ein einem weiteren Fenster angezeigt, das außerdem die Schaltfläche **Zur Zwischenablage** besitzt. Mittels dieser können Sie den Quellcode in die Zwischenablage kopieren und von dort in Ihre Applikation übertragen.

Die **Code generieren**-Schaltfläche gibt es derzeit nur in den Dialogen der **LayerDefinition** (**Layer** Register) und der **TableRow** (**TableRow** Register).

11.2.7 Eine Alt-Applikation editorfähig machen

Wenn Sie frühere (vor Version 3.0 erstellte) Applikationen mit dem neuen Eigenschaften-Editor bearbeiten wollen, erzeugen Sie zunächst aus der Alt-Applikation eine JGanttInitializer-Klasse und initialisieren damit Ihre Neu-Applikation.

Um den Eigenschaften-Editor aufrufen zu können, setzen Sie in den Quellcode Ihrer Initialisierungsfunktion den Aufruf

```
myJGantt.getPropertyEditor().setEnabled(true);
```

und führen das Programm aus.

Dann rufen Sie den Eigenschaften-Editor über die Tastenkombination <STRG> <Umschalt> <F10> oder mit dem Kontext-Menü aus Ihrer Alt-Applikation heraus auf (die sich in dem Zustand befindet, den Sie verwenden wollen). Nun rufen Sie durch Anklicken der entsprechenden Schaltfläche den Options-Dialog auf, deaktivieren die Checkbox **Generierten Code als Applikation mit Main-Methode speichern** und speichern mit **OK** ab.

Anschließend drücken Sie in der Editor-Oberfläche die Schaltfläche **Einstellungen speichern als** und speichern die Applikation als Initializer-Klasse in das Quellcodeverzeichnis ab, in welchem auch die ursprüngliche Klasse mit Main-Methode liegt.

Nun generieren Sie eine neue JGantt -Instanz und aktivieren die Initialisierung mit der neu abgespeicherten Initializer-Klasse:

```
myJGantt = new JGantt ();
new <Name der Initializer-Klasse>(myJGantt).initJGantt();
```

Die ursprünglichen JGantt-Setzungen bis zur dieser Stelle müssen Sie nun deaktivieren.

12 Spracheinstellungen

VARCHART JGantt unterstützt den Benutzer bei Interaktionen an vielen Stellen durch die Anzeige von textuellen Informationen in Form von Dialog- und Info-Fenstern. Bislang wurden solche Texte von VARCHART JGantt wahlweise auf Deutsch, Englisch oder Chinesisch angezeigt, je nach der Spracheinstellung des verwendeten Betriebssystems bzw. je nach der Setzung der Locale-Property der Java-Plattform. Diese Einstellung wird standardmäßig übernommen. Der Wert kann aber auch über die Java-API geändert werden:

```
java.util.Locale.setDefault(new java.util.Locale ("en", "EN"));
```

Sie können mit dem ersten Parameter die Sprache, mit dem zweiten das Land festlegen, der landestypische Eigenheiten der Sprache berücksichtigt. Das Beispiel oben setzt die Sprache auf "englisch" und das Land auf "England". Die Spracheinstellung sollten Sie am Beginn des Programms vornehmen, falls Sie eine Änderung der Vorgabe wünschen.

Die Spracheinstellung wirkt sich in VARCHART JGantt auf das Formatieren von Datumsausgaben (z.B. in der Zeitskala) und auf Texte in Menüs, in Informationsfenstern, in Editoren, Dialogen u. a. aus.

Das Formatieren der Datumsausgaben erfolgt in allen Sprachen, die die Laufzeitumgebung unterstützt. Texte in Menüs, Informationsfenstern, Editoren, Dialogen etc. können hingegen nur in deutsch, englisch und chinesisch ausgegeben werden. Beim Setzen der Parameters **de** oder **zh** erscheinen deutsche bzw. chinesische Texte, bei allen anderen Einstellungen erscheinen englische Texte.

Diese Texte können Sie bei Bedarf verändern und somit auch durch Texte in der gewünschten Sprache ersetzen. VARCHART JGantt kann um eigene Übersetzungen der Texte (lokalisierte Strings) erweitert werden. Dazu müssen zu den ResourceBundles weitere properties-Dateien mit den entsprechend übersetzten Texten in die JGantt.jar-Datei hinzugefügt werden.

12.1 Erstellung einer Properties-Datei

Eine Properties-Datei weist bestimmten Konstanten (z.B. in der Zeitskala) einen Wert zu, der verändert werden kann. Unten sehen Sie ein Beispiel, in

der die Konstanten der Zeitsakala mit Texten der englischen Sprache besetzt werden:

```
NE_TS_Modify_Section_Start_Date=Modify Section Start Date
NE_TS_Resize_Section_Width=Resize Section Width
NE_TS_SCALENAME_SECOND=second
NE_TS_SCALENAME_MINUTE=minute
NE_TS_SCALENAME_HOUR=hour
NE_TS_SCALENAME_6HOUR=6 hour
NE_TS_SCALENAME_DAY=day
NE_TS_SCALENAME_WEEKDAY=weekday
NE_TS_SCALENAME_WEEK=week
NE_TS_SCALENAME_CALENDARWEEK=calendarweek
NE_TS_SCALENAME_DAY+CALENDARWEEK=day+calendarweek
NE_TS_SCALENAME_WEEKDAY+CALENDARWEEK=weekday+calendarweek
NE_TS_SCALENAME_MONTH=month
NE_TS_SCALENAME_QUARTER=quarter
NE_TS_SCALENAME_HOUR_RELATIVE=relative hour
NE_TS_SCALENAME_DAY_RELATIVE=relative day
NE_TS_SCALENAME_DAY_DYNAMIC=dynamic
NE_TS_Next_Scale=zoom in
NE_TS_Previous_Scale=zoom out
```

Sie können die zugewiesenen Text-Werte in eine andere Sprache übersetzen und sich so eigene Sprachdateien erstellen. Unten sehen Sie die entsprechende chinesische Übersetzung mit Unicode-Zeichen:

```
NE_TS_Modify_Section_Start_Date=\u4fee\u6539\u5f00\u59cb\u533a\u57df\u65
e5\u671f
NE_TS_Resize_Section_Width=\u8c03\u6574\u9009\u62e9\u533a\u57df\u5bbd\u5
ea6
NE_TS_SCALENAME_HOUR=\u5c0f\u65f6
NE_TS_SCALENAME_6HOUR=6 \u5c0f\u65f6
NE_TS_SCALENAME_DAY=\u5929
NE_TS_SCALENAME_WEEKDAY=\u5468\u65e5
NE_TS_SCALENAME_WEEK=\u5468
NE_TS_SCALENAME_CALENDARWEEK=\u65e5\u5386\u5468
NE_TS_SCALENAME_DAY+CALENDARWEEK=\u5929+\u65e5\u5386\u5468
NE_TS_SCALENAME_WEEKDAY+CALENDARWEEK=\u5468\u65e5+\u65e5\u5386\u5468
NE_TS_SCALENAME_MONTH=\u6708
NE_TS_SCALENAME_QUARTER=\u5b63\u5ea6\uff08\u523b\uff09
NE_TS_SCALENAME_HOUR_RELATIVE=\u76f8\u5bf9\u5c0f\u65f6
NE_TS_SCALENAME_DAY_RELATIVE=\u76f8\u5bf9\u5929
NE_TS_SCALENAME_DAY_DYNAMIC=\u52a8\u6001
NE_TS_Next_Scale=\u653e\u5927
NE_TS_Previous_Scale=\u7f29\u5c0f
```

12.2 Verzeichnisse der Properties-Dateien

Die Sprachdateien werden unter den unten aufgelisteten Namen und in den entsprechenden Verzeichnissen bzw. in die jar-Dateien (die ja komprimierte

Datei-Sammlungen darstellen) abgespeichert. Der Namensteil „xx“ steht dabei für die in Java übliche Länderkennung (z.B. fr für Frankreich).

Texte der Diagramm-Schaltleiste (Diagram Control Panel)**:**
de\netronic\bean\diagramcontrolpanel\NeDiagramControlPanel-Res_xx.properties

Texte des "Daten editieren"-Dialogs:
de\netronic\bean\entityeditor\NeEntityEditorRes_xx.properties

Info-Texte bei Interaktionen im Gantt-Graph, z.B. beim Verschieben von Knoten oder Layern:
de\netronic\bean\ganttgraph\NeGanttGraphRes_xx.properties

Texte des Kontext-Menüs im Gantt-Graph:
de\netronic\bean\layouter\NeLayouterRes_xx.properties

Texte bei Interaktionen in der numerischen Skala bei Histogrammen:
de\netronic\bean\numericscale\NeNumericScaleRes_xx.properties

Tooltip-Texte in der Druckvorschau:
de\netronic\bean\printmanager\NePrintManagerRes_xx.properties

Texte bei Interaktionen in der Zeitskala:
de\netronic\bean\timescale\NeTimeScaleRes_xx.properties

Texte in der Infobox mit Lizenzierungsinformationen:
de\netronic\common\beanbase\NeBoxMessageRes_xx.properties

Texte des Fontauswahl-Dialogs und weitere allgemeine Texte:
de\netronic\common\beanbase\NeCommonBeanbaseRes_xx.properties

Infotexte bei Interaktionen im Gantt-Graph, wenn ValueReferences eingesetzt worden sind:
de\netronic\common\beanbase\NeValueReferenceRes_xx.properties

13 Drucken mit VARCHART JGantt

VARCHART JGantt zeichnet sich durch sehr komfortable Druckmöglichkeiten aus. Vor dem Ausdruck bietet die Komponente dem Benutzer eine Palette von Optionen, die auch hohen Ansprüche beim Ausdruck von Gantt-Diagrammen genügen.

13.1 Der Umfang der Druckfunktionalität

Der Benutzer kann:

- das Diagramm skalieren
- es auf Blätter aufteilen
- festlegen, auf wie viele Seiten in Höhe oder Breite das Diagramm partitioniert werden soll
- jedes einzelne Blatt im Detail ansehen und dabei bequem navigieren
- eine Kopfzeile anlegen
- eine Fußzeile anlegen
- diese unabhängig voneinander positionieren
- Seitenzahlen setzen
- Druckdatum einfügen
- Schriftart, -grad und -schnitt für Kopf- und Fußzeile festlegen
- Zeitbereiche des Diagramms definieren und nur diese drucken

Das Interface **JGIPrintmanager** der Programmierschnittstelle enthält Eigenschaften und Methoden, die die Druckfunktionen von VARCHART JGantt ansteuern. Sie benötigen die manuelle Ansteuerung jedoch nur, wenn Sie z.B. Voreinstellungen ändern wollen, die beim Aufruf des Programms wirksam sein sollen.

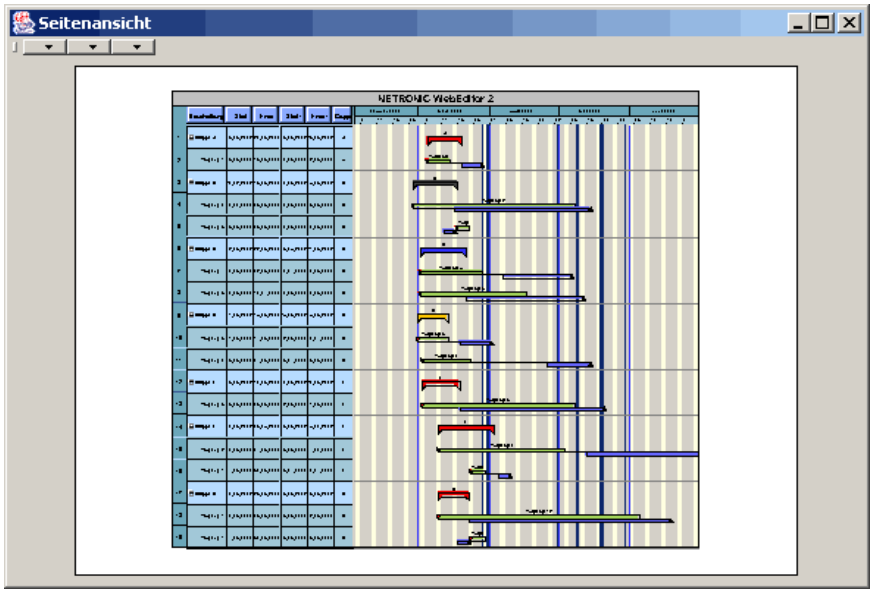
Wir stellen Ihnen im Folgenden die Druckfunktionen und ihre Ansteuerungsmöglichkeiten vor. Über das Beispielprogramm jganttsample.jar, das Bestandteil der Auslieferung ist, können Sie die nachfolgend gezeigten Funktionen ausprobieren.

Über die Schaltfläche Seitenansicht ...

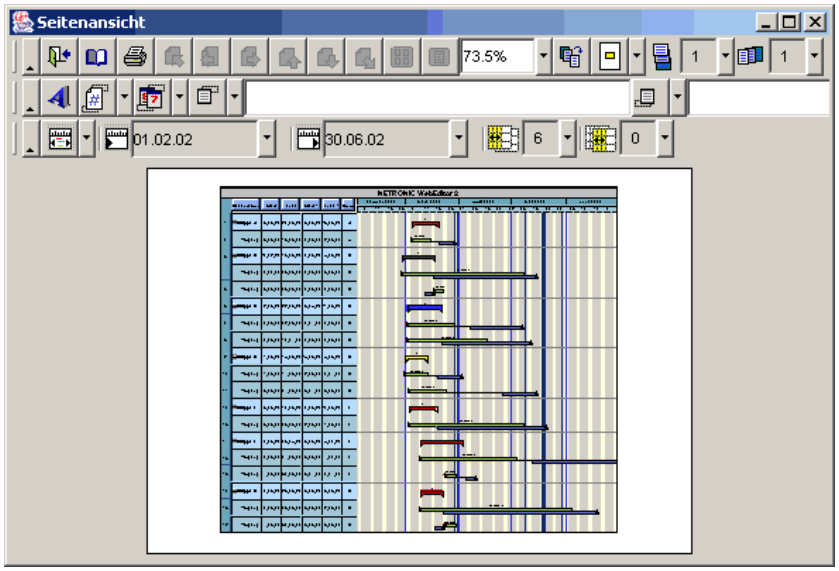
... gelangen Sie zum entsprechenden Fenster.

13.2 Funktionsleisten der Seitenansicht

Die Seitenansicht bietet dem Benutzer drei verschiedene Funktionsleisten an, die er beliebig ein- und abschalten kann. In der Grafik unten sehen Sie die Schaltflächen in der linken oberen Ecke:



Nach dem Zuschalten werden die drei Funktionsleisten sichtbar:



Standardmäßig sind die beiden unteren der drei Schaltleisten beim Programmaufruf abgeschaltet. Sie können diese jedoch per API anschalten:

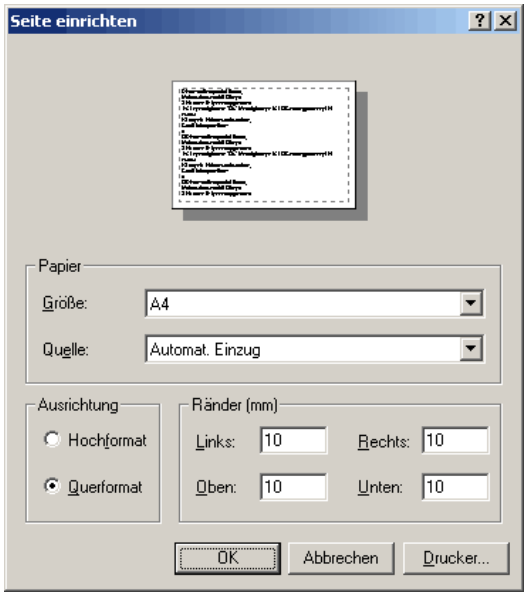
```
jGantt1.getPrintManager().setHeaderFooterToolbarVisible(true);
jGantt1.getPrintManager().setGanttOptionsToolbarVisible(true);
```

Die obere Schaltleiste enthält zwei Funktionsbereiche: den Bereich mit den ersten drei Schaltflächen für seitenvorschaufremde Funktionen, sowie den Bereich dahinter für die eigentlichen Seitenvorschau-Funktionen wie Seitennavigation, Seitenaufteilung und Skalierung.

Mit der ersten Schaltfläche im linken Bereich schließen Sie die Seitenvorschau:

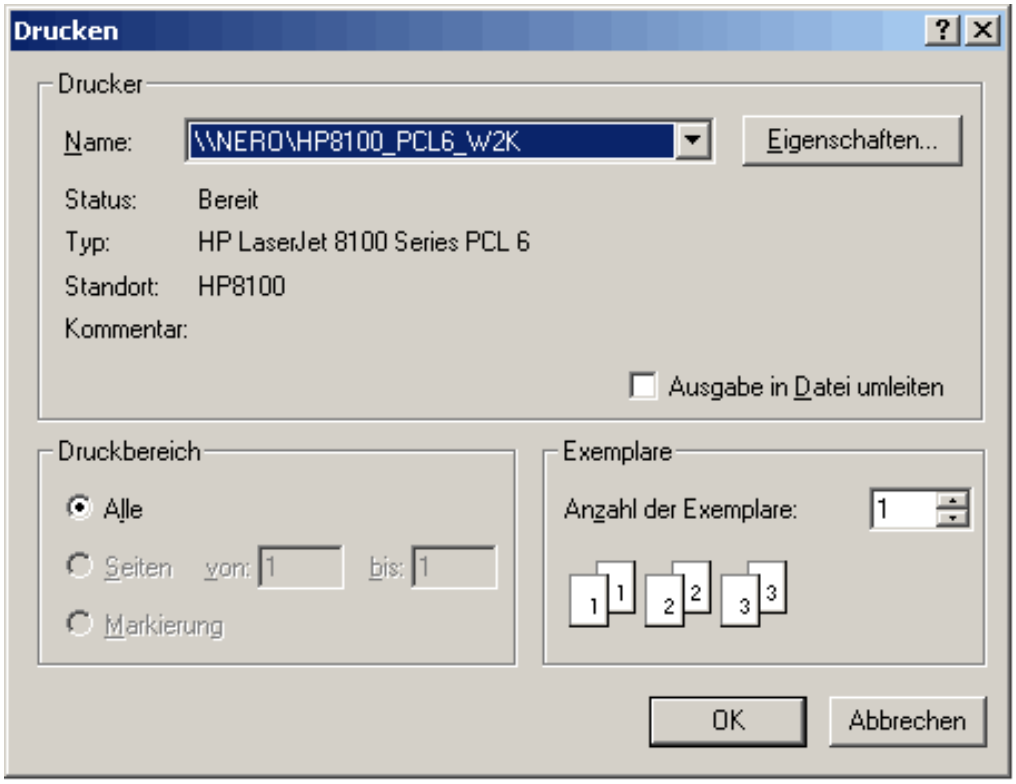
Über die nächste Schaltfläche können Sie den Systemdialog Ihres Druckers aufrufen und dort die druckerspezifischen Festlegungen für die Druckseiten eingeben:

Nach dem Anklicken erscheint ein System-Dialog, z.B. von Windows:



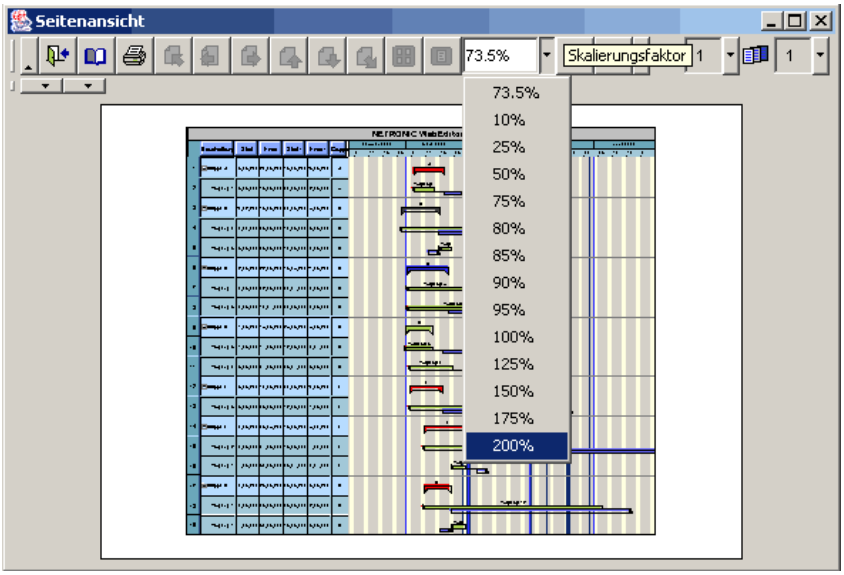
Mit der dritten Taste rufen Sie den Druckdialog für das Diagramm auf:

Es erscheint der Druckdialog der Systemplattform, hier z.B. von Windows:

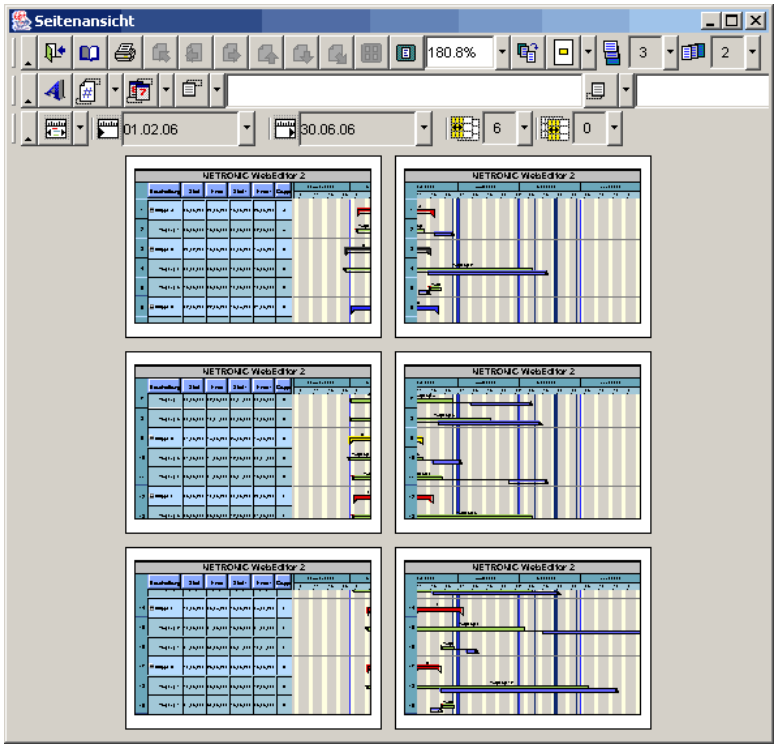


Mit den folgenden Tasten und Feldern rechts davon können Sie das Diagramm vor dem Druck in optimaler Weise an Ihre Wünsche anpassen.

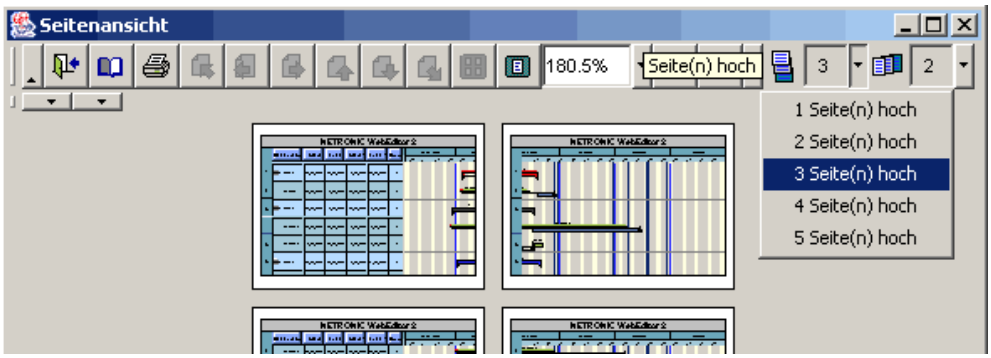
Mit der Skalierung geben Sie vor, wie groß die Grafik werden soll (Angaben in Prozent). Bei einer Veränderung der Skalierung teilt die Seitenvorschau das Diagramm automatisch auf die entsprechende Anzahl von Seiten auf, so dass das Diagramm optimal an die vorgegebene Seitenzahl angepasst wird. VARCHART JGantt verhindert dabei automatisch, dass die Tabelle des Diagramms mehr als eine Seitenbreite beansprucht. Der Skalierungsfaktor wird ggf. dementsprechend reduziert. Im Beispiel unten wird eine Vergrößerung von 200% gewählt:



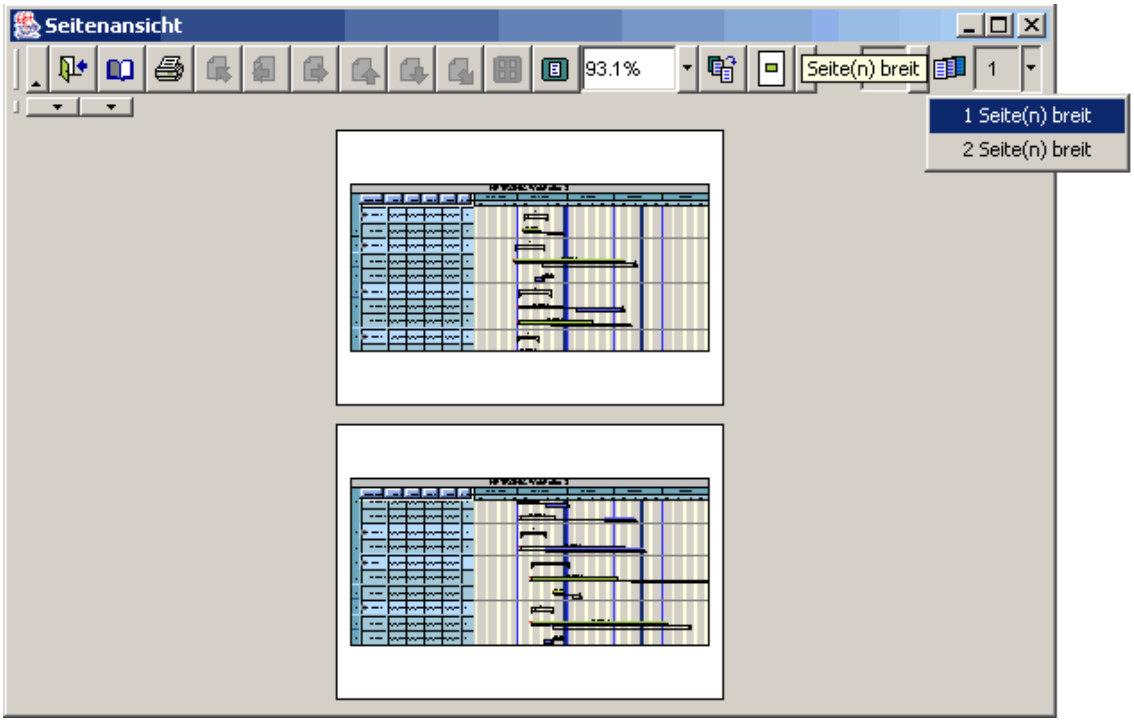
Die nachfolgende automatische Optimierung ergibt einen erlaubten Skalierungswert von 180,8% und das Diagramm wird automatisch auf 6 Seiten verteilt:



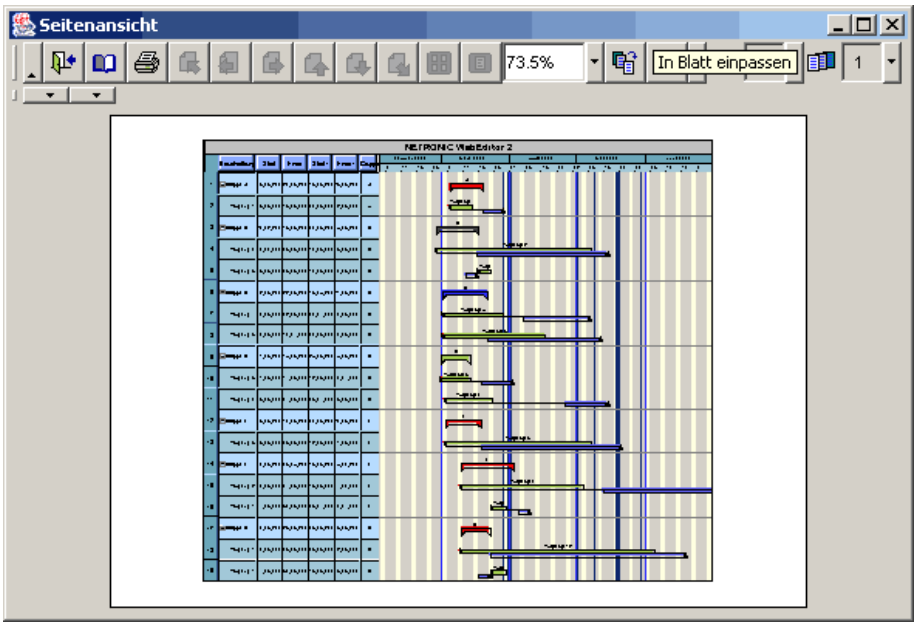
In den Feldern außen rechts wird Ihnen angezeigt, über wie viele Seiten in der Vertikalen und in der Horizontalen sich das Diagramm aufteilt:



Diese Werte können Sie ändern. Um Verzerrungen des Diagramms zu vermeiden, wird bei einer Veränderung der horizontalen Seitenzahl die Anzahl der vertikalen Seiten automatisch mitgeändert und umgekehrt. Der entsprechende Skalierungsfaktor wird angezeigt. Im Beispiel unten haben wir die Anzahl der horizontalen Seiten auf 1 reduziert. Die Anzahl der vertikalen Seiten wird automatisch auf 2 gesetzt und der Skalierungsfaktor ergibt 93,1 %:

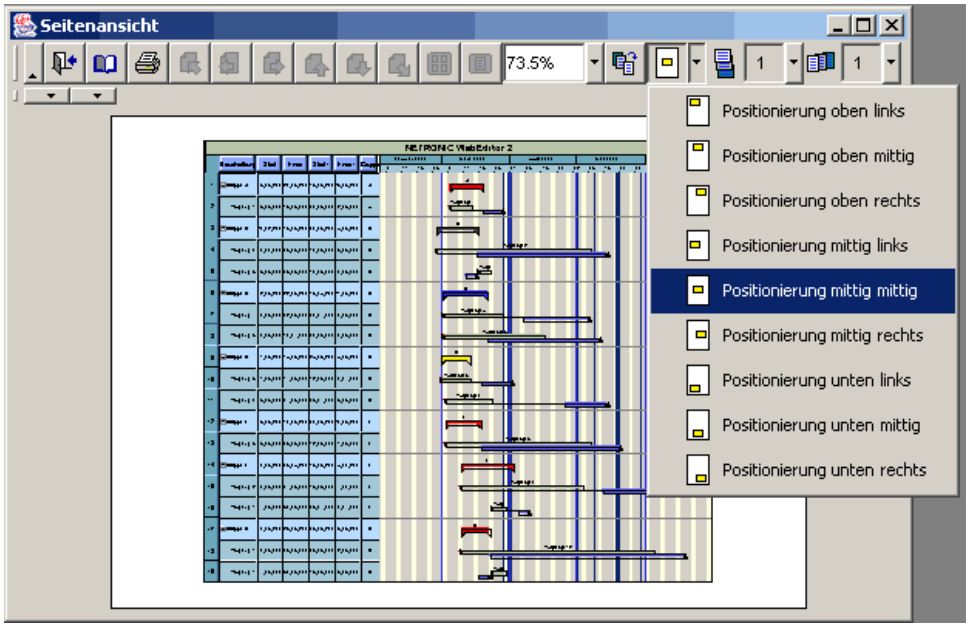


Sie können ein auf mehrere Seiten verteiltes Diagramm über die Schaltfläche rechts neben der Skalierung schnell und optimal auf 1 Blatt einpassen, wobei ebenfalls der Skalierungsfaktor automatisch neu berechnet wird:



Sie können das Diagramm bzw. den Diagrammausschnitt innerhalb einer Seite positionieren, unabhängig davon, ob sich das Diagramm auf eine oder auf mehrere Seiten verteilt. Dazu benutzen Sie die Schaltfläche mit dem Positio-

nierungssymbol. Das Menü bietet Ihnen eine Auswahl von Positionierungsoptionen an:



Auch diese Eigenschaft ist per API ansteuerbar:

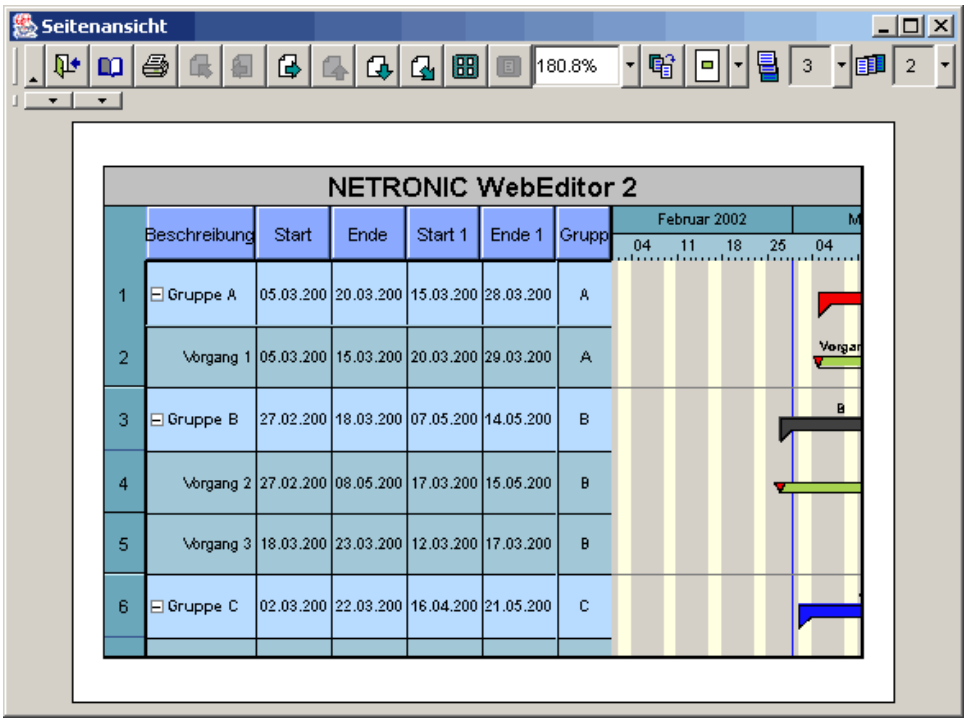
```
jGantt1.getPrintManager().setAlignment(ALIGNMENT_CENTER_CENTER);
```

Wenn Sie ein Diagramm auf mehrere Seiten verteilt haben, bietet VARCHART JGantt Ihnen die Möglichkeit, entweder alle Seiten im Überblick oder jede Seite einzeln anzuschauen. Wenn Sie sich im Modus **Alle Seiten** befinden, bietet Ihnen die Seitenvorschau die Möglichkeit, in den Einzelseiten-Modus umzuschalten. Die entsprechende Schaltfläche erscheint dann aktiviert:

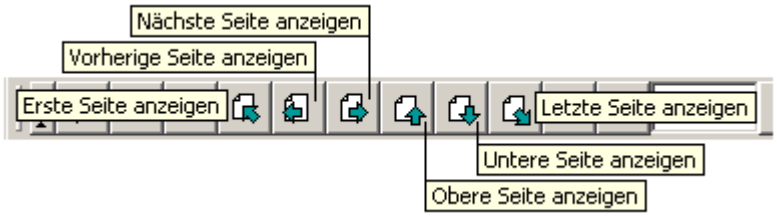
Wenn Sie diese Schaltfläche drücken, geschehen drei Dinge:

1. Sie können vom Einzelseiten- in den "Alle-Seiten"-Modus zurückwechseln. Dazu erscheint die komplementäre Taste aktiviert:

2. Die erste Teilseite (Einzelseite) wird angezeigt.



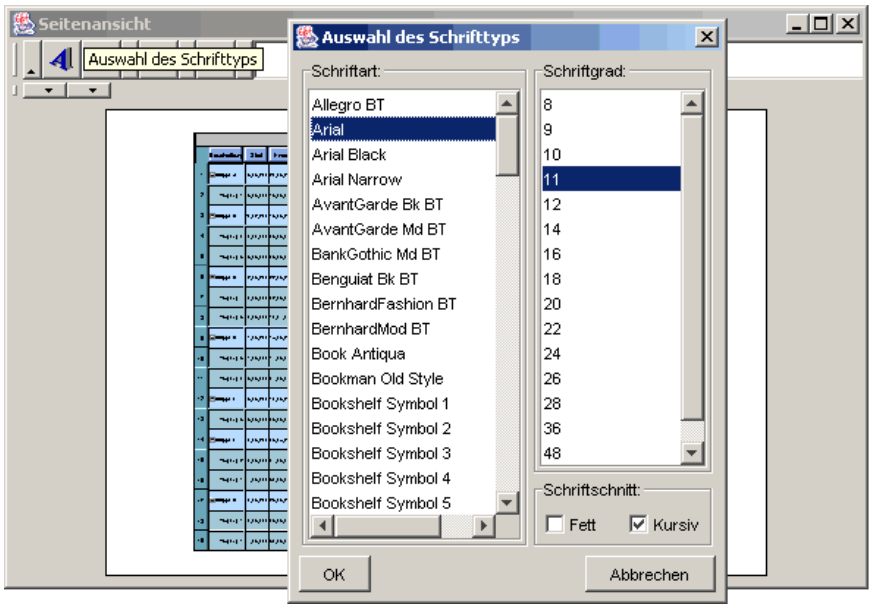
3. Die Schaltflächenbereich zum Navigieren durch die Einzelseiten erscheint aktiviert.



Mit diesen Tasten können Sie zur ersten oder letzten Seite springen sowie relativ zur aktuellen Seite nach links, rechts, oben oder unten wechseln. Falls eine der Möglichkeiten nicht existiert, z.B. das Navigieren nach links bei der aktuellen Position auf der ersten Seite, erscheint die entsprechende Taste automatisch deaktiviert.

Die mittlere Schaltleiste enthält Funktionen zur Gestaltung der Kopf- und Fußzeile:

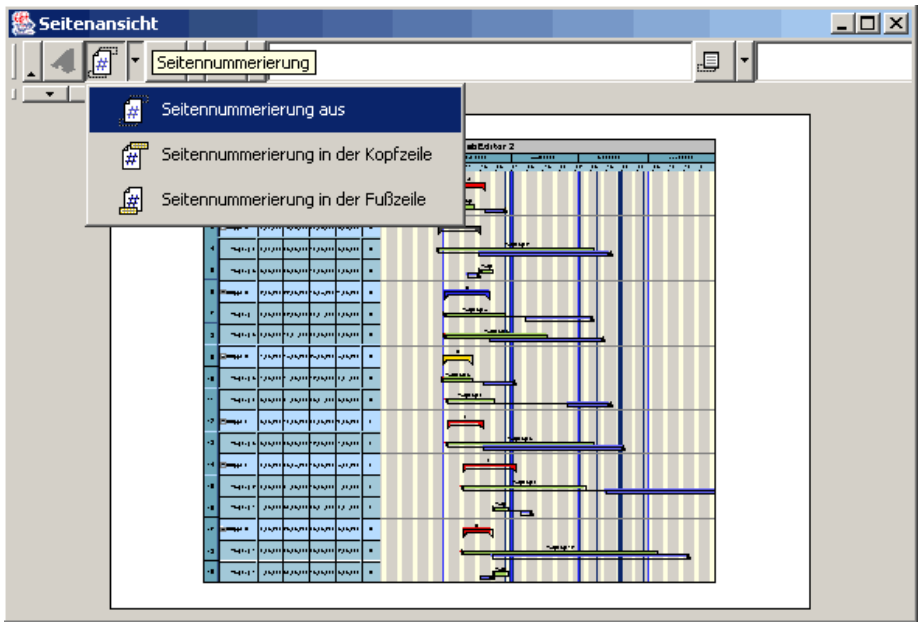
Über die erste Schaltfläche nehmen Sie Einstellungen der Schriftart, des Schriftgrades und des Schriftschnitts in Kopf- und Fußzeile vor:



Diese Eigenschaft steht Ihnen auch in der API zur Verfügung:

```
jGantt1.getPrintManager().
   setHeaderFooterFont(new java.awt.Font("SansSerif ", 0, 8));
```

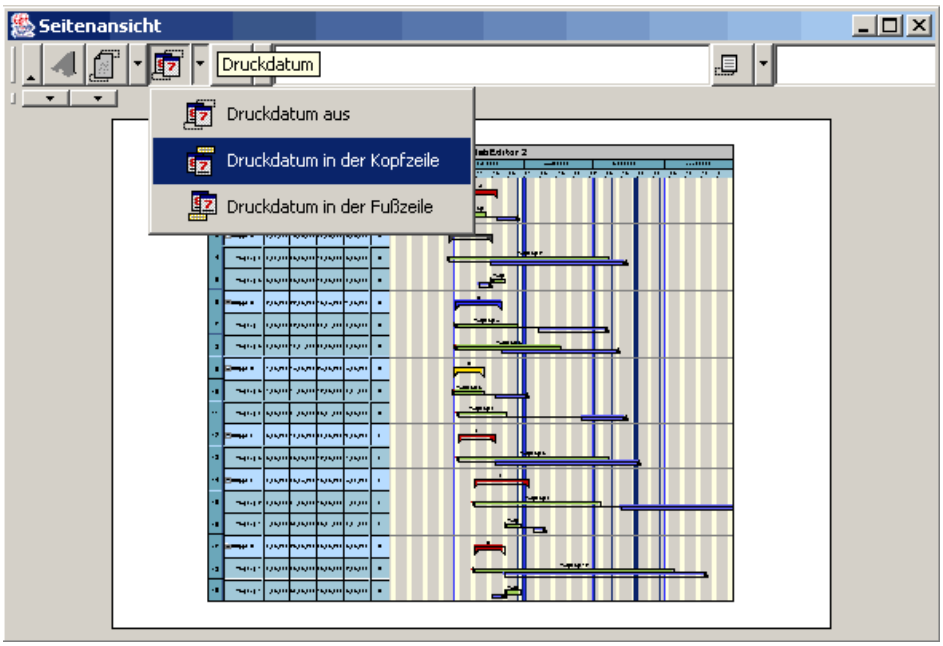
Mit der zweiten Schaltfläche stellen Sie ein, ob eine Seitennummerierung eingefügt werden soll, und falls ja, ob in der Kopf- oder Fußzeile:



Per API:

```
jGantt1.getPrintManager().setPageNumbersPosition
(JGIPrintManager.PAGE_NUMBERS_OFF);
```

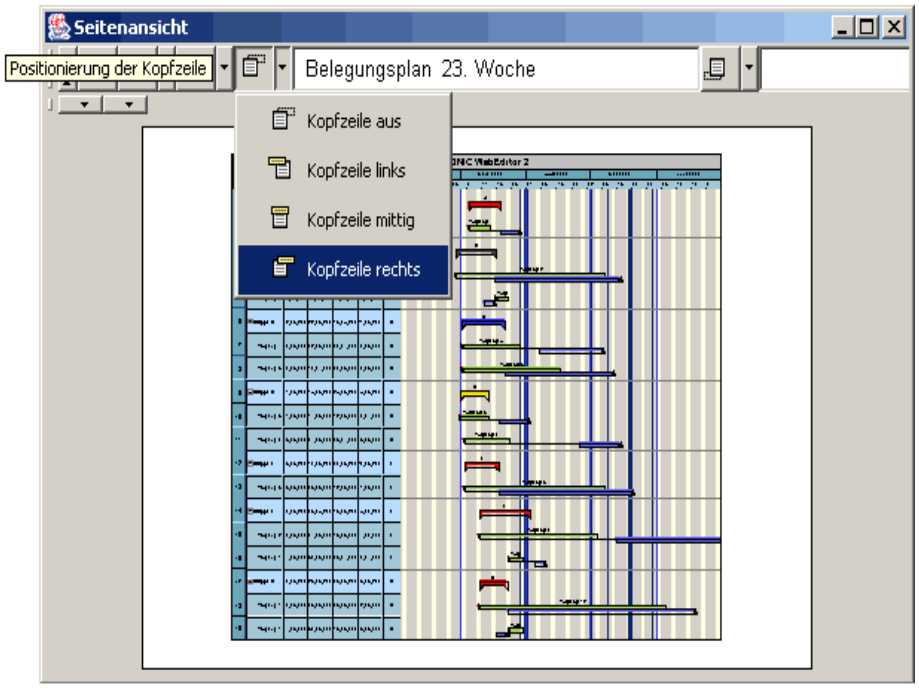
Mit der nächsten Taste können Sie das aktuelle Datum als Druckdatum an- oder abschalten und es in die Kopf- oder Fußzeile setzen.



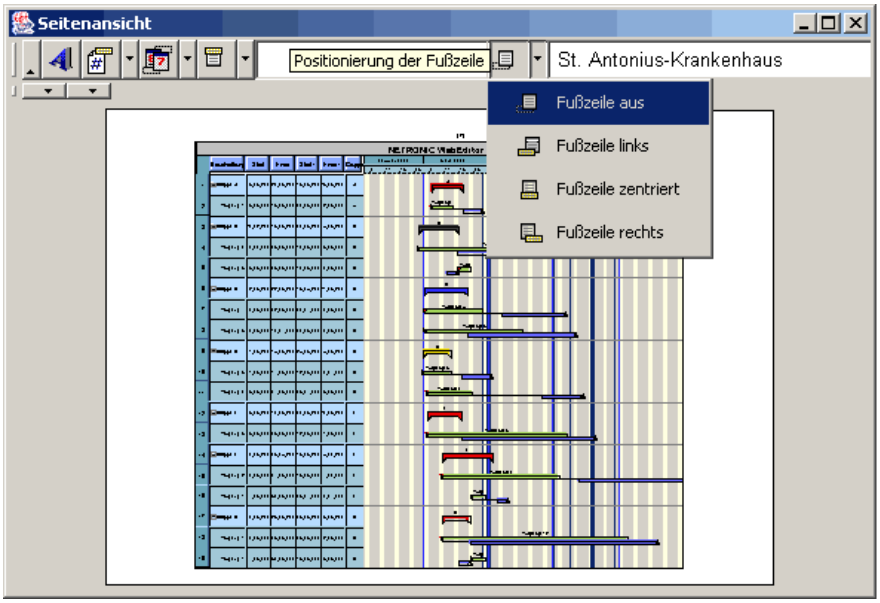
Per API:

```
jGantt1.getPrintManager().setPrintDatePosition
(JGIPrintManager.PRINT_DATE_INTO_HEADER);
```

Mit der nächsten Taste und dem danebenliegenden Feld können Sie den Text der Kopfzeile editieren, die Kopfleiste an- oder ausschalten und sie positionieren:



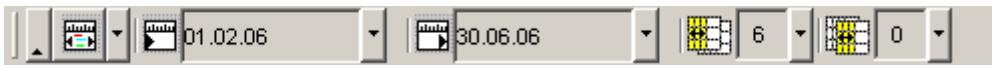
Entsprechende Optionen werden Ihnen für die Fußzeile angeboten:



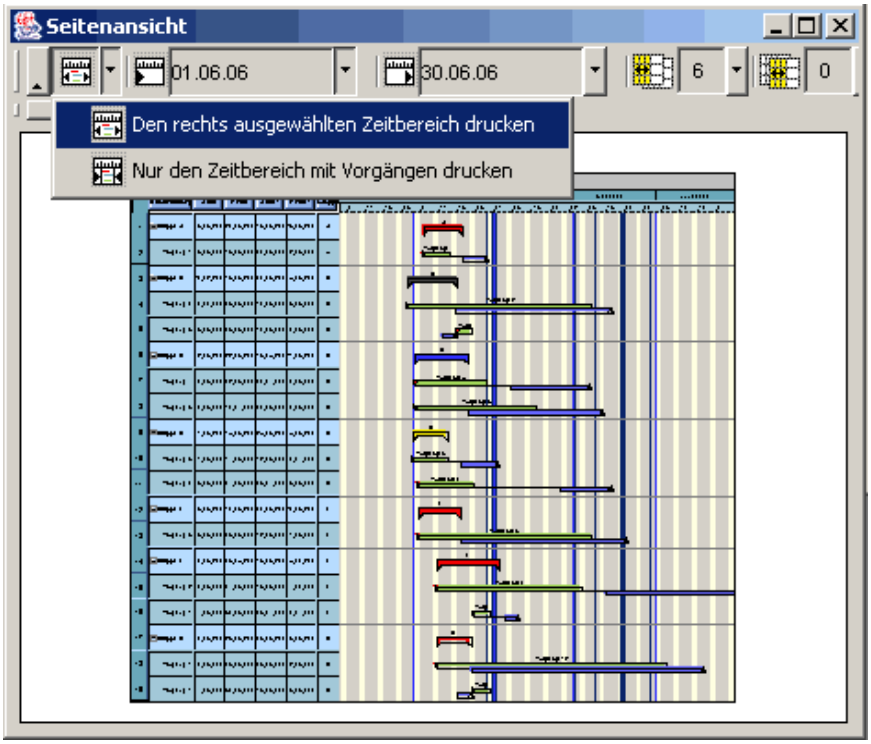
Die Ausrichtung sowie Betextung von Kopf- und Fußzeile können Sie auch per API ansteuern:

```
jGantt1.getPrintManager().setHeaderText("Belegungsplan 23. Woche");
jGantt1.getPrintManager().setFooterText("St. Antonius-Krankenhaus");
jGantt1.getPrintManager().setHeaderAlignment(HEADER_RIGHT);
jGantt1.getPrintManager().setFooterAlignment(FOOTER_LEFT);
```

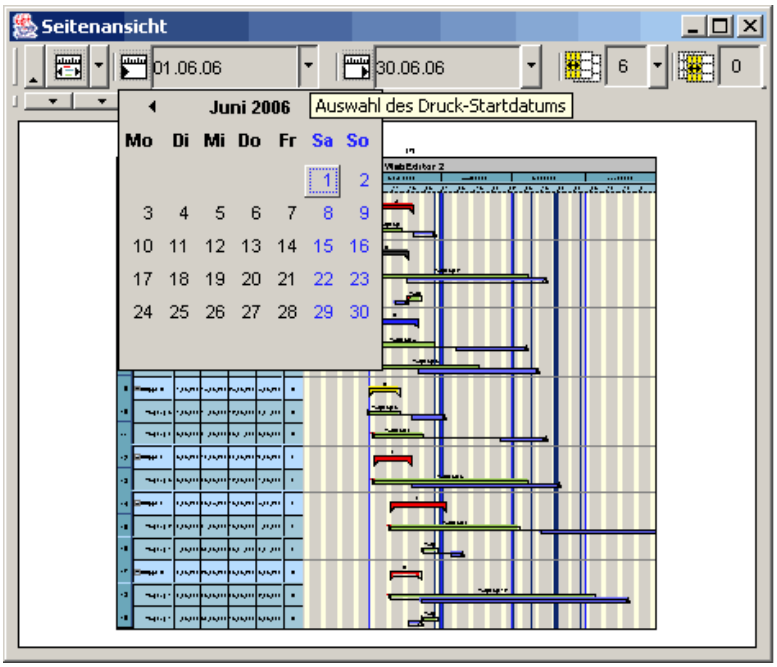
Mit der unteren Schaltleiste können Sie Zeitbereiche im Diagramm für den Druck definieren:



Mit der linken Schaltfläche können Sie festlegen, ob Sie einen über das Start- und Enddatum zu definierenden Bereich drucken wollen oder automatisch nur die Zeitspanne aussuchen lassen, in der sich Vorgänge befinden:



In den Feldern daneben können Sie das Anfangs- und Enddatum für den zu druckenden Bereich festlegen:



Für die Ansteuerung per API legen Sie Anfang und Ende des zu druckenden Bereichs fest (in Millisekunden):

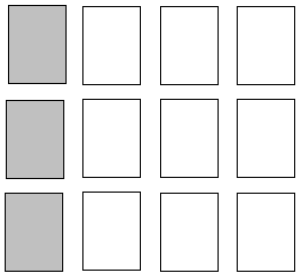
```
jGantt1.getPrintManager().setPrintStartTime(1136156400000L);
```

```
jGantt1.getPrintManager().setPrintEndTime(1138834800000L);
```

…oder Sie drucken nur den Bereich, in dem sich Knoten befinden (Anfangs- und Enddatum werden automatisch ermittelt):

```
jGantt1.getPrintManager().setPrintSectionNodesOnly(true);
```

VARCHART JGantt bietet Ihnen die Möglichkeit, die Anzahl der Tabellenspalten anzugeben, die gedruckt werden sollen. Wenn ein Diagramm auf eine Matrix von Blättern gedruckt wird, kann diese Zahl für die am weitesten links befindlichen Seiten (Haupt- oder Deckseiten) eine andere sein als für die Folgeseiten (Wiederholungsseiten).



Grau: Haupt- oder Deckseiten, weiß: Folge- oder Wiederholungsseiten

Die Programmierschnittstelle stellt Ihnen dazu folgende Eigenschaften zur Verfügung.

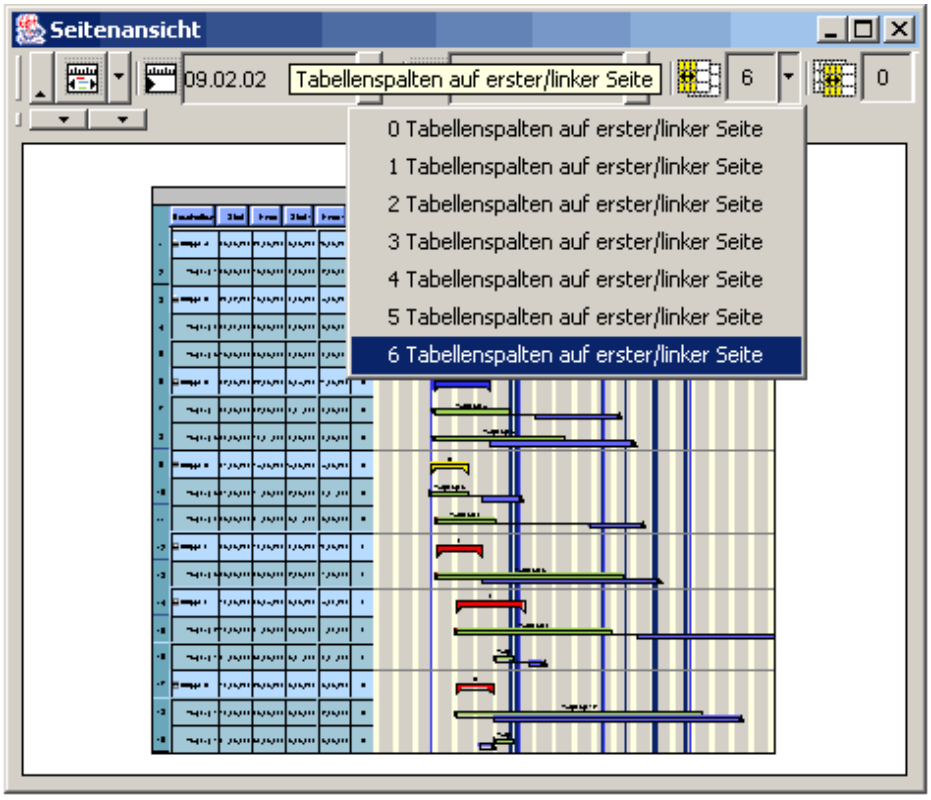
Für die Deckblätter:

```
jGantt1.getPrintManager().setMainTableColumns(6);
```

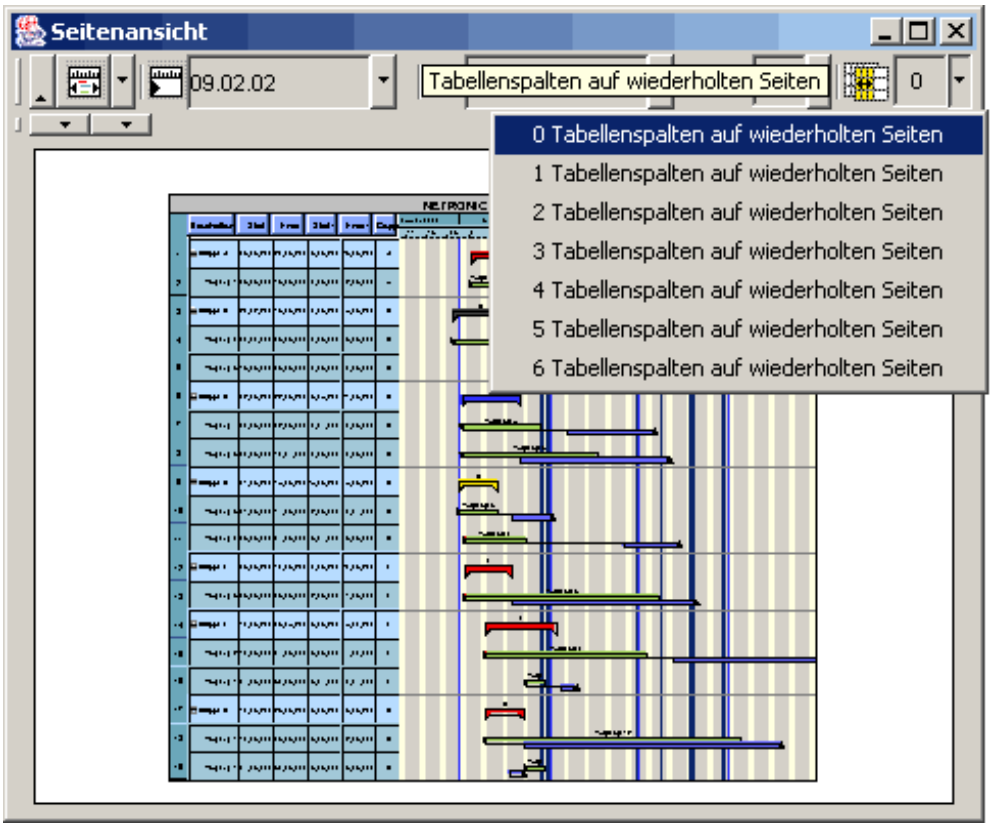
Für die Widerholungsseiten:

```
jGantt1.getPrintManager().setRepeatedTableColumns(0);
```

In der Schaltleiste dient das zweitletzte Feld der Angabe der Tabellenspalten auf den Hauptseiten:



Im letzten Feld geben Sie die Anzahl der Tabellenspalten für die Folgeseiten vor:



Damit haben Sie die komfortabeln Druckmöglichkeiten von VARCHART JGantt kennengelernt. Sie können im Referenzhandbuch beim Interface JGIPrintManager weitere Eigenschaften für die Programmierschnittstelle nachschlagen.

14 Index

A

Aktion 159, 161, 163, 164
Ereignis 160
AppData 24, 25, 53
Applikationsdaten 24, 53
Arbeitszeit 112
Asterisk 118
Attribut 23, 26, 53, 55
Auslastung 139

B

Benutzerinteraktion 147
Benutzers, Sichtweise des 12
Bereichswert 124, 125
Bewegungsmodus 148
Bildlauf
automatischer 147
Bitmap 80
natürliche Größe 81
Blattknoten
als Ziel einer Verbindung 149

C

Customizer 24, 25, 43

D

Daten 59
Anlage 57
Datendefinition 53, 58
Datenstruktur 23, 25, 28
Datenstrukturen 52, 53, 59
über API 52
über XML 52
Dekorations-Objekt 77, 83
Diagramm
Schaltleiste 155, 156, 158
Teilbereiche expandieren 99
Teilbereiche kollabieren 99
Drag&Drop 148
Drucken 189, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 204
Funktionsleisten 190
Dynamische Farben 123

E

Eigenschaften-Browser 173
Eigenschaften-Editor 12, 176, 178, 181
Alt-Applikation 183
Neuapplikation 181
Schnappschuss 181
selbstgeschriebene Klassen 179
Einrückung, hierarchische 110
Ende-Ende-Beziehung 135
Ende-Start-Beziehung 135
Entität 23, 26, 53, 59
Änderung 60
Neuanlage 60
Entitäten-Set 23, 25, 28, 53, 54, 55, 58
Änderungen 59
Ereignisse 59
EventListener 59, 60

F

Farbe
dynamisch 123, 124, 125, 126

Farbe, alternierende Zeilen- 70
Farbe, dynamische Zeilen- 70
Farbe, rotierende Zeilen- 71
Farbe, Zuordnungs- 124, 127
Farbschema 127
alternateColor 20
alternateTExtColor 20
lineColor 20
mainColor 20
shadedAlternateColor 20
shadedAlternateTextColor 20
shadedColor 20
shadedTextColor 20
textColor 20
Farbverlauf
über Zeilen 72
Farbzuordnungstabelle 124
Feiertag 117, 120
beweglicher 118
Füllmodus 78, 79, 80, 81

G

Gantt-Diagramm 8, 63
Gantt-Graph 8, 40
alternateColor 41
Farbschema 40
mainColor 41
shadedAlternateColor 41
shadedColor 41
Gitter 111
Farbe 112
Gitter, Kalender- 111, 112, 115, 116, 117, 118, 128
Gitter, Linien- 111
Gruppe
Layout 94
Sortierung 98
Gruppenknoten 92, 93, 94, 95, 97, 98
automatisch anpassen 98
automatisch erstellen 99
Farbschema 94
Gruppierungsebene 97

H

Hierarchie 102, 104, 105, 107
Einrückung 110
Minuszeichen 100
Pluszeichen 100
Programmierung 108
Rang 106
Teilbaum 105
Umstufung 102
Hierarchie-Code 99, 108, 109
Hierarchie-Ebene 100
hierarchische Strukturen 99
Histogramm 139
arbeitsfreie Zeiten 146
Grenzwertlinie 143
Kalender 145
Kurve 140, 142, 145

I

Informationsfenster
unterdrücken 149
Interaktion 147
Anlegemodus 14
Ereignisse 150
Feld verbreitern 13
Knoten anlegen 13
Knoten löschen 14
Stufe 1 151, 152
Stufe 2 151, 153
Stufe 3 151, 154
Stufe 4 151, 154

Zeitskala verbreitern 13

J

JGantt-Instanz 16

K

Kalender 113
Persistenz 122, 123
Kalendergitter 111
Knoten 23, 74, 76
Anlegemodus 150
Beschriftung 36
Dekoration 76, 77
Design 30, 74
Farbgebung 29
Farbschema 29, 94
Fortschrittsbalken 74
graphische Struktur 76
Gruppierung 89, 90, 91, 93, 94
Hierarchie 89, 90, 91, 94, 99
Layer 76
Layer-Definition 76
ohne Ausdehnung 35
Selektiermodus 150
Sortierung 87, 88
Sprunggröße beim Verschieben 150
Symbol 34
Termin, absoluter 74
Termin, relativer 74
Termine 74
Veränderbarkeit der Größe 148
Knotenbeschriftung 78
Datumsformat 39
Position 38
Schrifteigenschaften 38
Knoten-Design
3Rects 32
LineCompletionRect 32
LineRect 31
Ramps 33
Rect 31
summary 33
Symbol 31
Knotenphantom 40
Positionierungslinie 40
Knotentermine 26
Komponenten 11
Kontextmenü
unterdrücken 149
Kurve 139
Daten 142
Flächenbildung 139

L

Label-Objekt 77, 78
Layer 76
Anfangsdatum 76
Enddatum 76
Typ 77
ValueReference-Objekt 76
Linienkurve 139
lokalisierte Strings 185

M

Master-Instanz 170
Mehrfachgruppierung 97

N

Nichtarbeitszeit 112

P

Picture-Objekt 77, 78, 79, 81
bevorzugte Größe 77
Profil 112, 114, 115, 116, 119, 120, 121
Bestückung von 117

Profil, Display- 116
Profil, Knoten- 116, 121
Property Browser 173
Pufferzeit
freie 165
Gesamt- 165

S

Scheduler 166
Slave-Instanz 170
Spaltenüberschrift 27
Spracheinstellungen 185
Start-Ende-Beziehung 135
Start-Start-Beziehung 135
Stufenkurve 139
Symbol 78
Symbol-Objekt 82
Synchronizer-Panel 169

T

Tabelle 10, 42, 63
alternateColor 44
Anzahl Spalten 66
Datenquelle für Spalten 66
Datumsformat 66
Editierbarkeit 66
Einrückung nach Hierarchie 66
Farbe 66
Farbschema 44
Felddefinition 67
Felder über mehrere Spalten 66
Feldstil 68
Gestaltung 64
Inhaltsdefinition 66
Kern 63
mainColor 44
Optimale Spaltenbreite 69
Schriftart Basiszeilen 66
Schriftart Gruppenzeilen 66
Schriftart Spaltentitel 66
Schriftart Zeilentitel 66
Schrifteigenschaften 45
Schrifteigenschaften Spaltenüberschrift 45
Schrifteigenschaften Tabellengruppe 45
Schrifteigenschaften Zeilenüberschrift 45
shadedAlternateColor 44
shadedColor 44
Sichtbarkeit 66
Sichtbarkeit Spaltentitel 66
Spalten 42, 63
Spaltenbreite 66
Spaltentitel 63
Spaltenüberschriften 44
Standardzeilendefinition 67
Zeilen 42, 63
Zeilendefinition 64, 67
Zeilentitel 63
Zeilentyp 64, 67
Terminlinie 41

V

ValueReference-Objekt 76, 84
Verbindung 135, 136, 138
Anlege-Modus 136
Ansatztermin 138
Auswahl-Modus 137
Endpunkt 137
Startpunkt 137
Termine 138
Typ 135, 137
Voraussetzungen, technische 10

X

XML-Datei 55, 56

Z

Zeitberechnung 165, 166, 167
zeitliches Muster 111, 117
Zeitrechner 166
Zeitskala 8, 17, 111
3D-Effekt 19
Anfangsdatum 17
Auflösung 18
Ausdehnung 17
Enddatum 17
Farbschema 20
Position 19
Schriftarten 22
Typen 18
ViewStart 17
Zeitspanne 117, 119, 120
einmalig 117
wiederkehrend 117
Zuordnungsfarbe 127
Zuordnungstabelle 127